Microsoft® DirectX™ 2
Software Development Kit

C H A P T E R 2

DirectDraw

Overview

About DirectDraw

DirectDraw™ is a DirectX™ 2 SDK component that allows direct manipulation of display memory, hardware blitters, hardware overlays, and page flipping. DirectDraw provides this functionality while maintaining compatibility with existing Microsoft® Windows® 95 applications and device drivers.

The DirectX 2 SDK allows you an unprecedented level of access to display and audio hardware, while it insulates you from the specific details of that hardware. DirectX 2 is built for speed. In keeping with these goals, DirectDraw is not a high-level graphics application programming interface (API).

DirectDraw for Windows 95 is a software interface that provides direct access to display devices while maintaining compatibility with the Windows graphics device interface (GDI). DirectDraw provides a device-independent way for games and Windows subsystem software, such as 3D graphics packages and digital video codecs, to gain access to display device-dependent features.

DirectDraw works with a wide variety of display hardware, ranging from simple SVGAs to advanced hardware implementations that provide clipping, stretching, and non-RGB color format support. The interface is designed so that your applications can request the capabilities of the underlying hardware, then use those capabilities as required.

DirectDraw provides the following display device-dependent benefits:

- Supports double-buffered and page flipping graphics.
- Provides access to, and control of, the display card's blitter.
- Supports 3D z-buffers.
- Supports hardware assisted overlays with z-ordering.
- Improves graphics quality by providing access to image-stretching hardware.
- Provides simultaneous access to standard and enhanced display device memory areas.

DirectDraw's mission is to provide device-dependent access to display memory in a device-independent way. Your application need only recognize some basic device dependencies that are standard across hardware implementations, such as RGB and YUV color formats and the stride between raster lines. You need not worry about the specific calling procedures required to utilize the blitter or manipulate palette registers. Essentially, DirectDraw manages display memory. Using DirectDraw, you can manipulate display memory with ease, taking full advantage of the blitting and color decompression capabilities of different types of display hardware without becoming dependent on a particular piece of hardware.

DirectDraw provides world-class game graphics on computers running Windows 95 and Windows NT®.

Introduction to DirectDraw

DirectDraw provides display-memory and display-hardware management services. It also provides the usual functionality associated with memory management: memory can be allocated, moved, transformed, and freed. This memory represents visual images and is referred to as a surface. Through the DirectDraw hardware abstraction layer (HAL), applications are exposed to unique display hardware functionality, including stretching, overlaying, texture mapping, rotating, and mirroring.

Architectural Overview

DirectDraw

DirectDraw is the Windows system component that performs the common functions required by both hardware and software implementations of DirectDraw. As such, DirectDraw is the only client of the DirectDraw hardware abstraction layer (HAL). Applications must write to DirectDraw. DirectDraw returns two sets of capabilities, one for hardware capabilities and one for software emulation capabilities. Using these, an application can easily determine what DirectDraw is emulating and what functionality is provided in hardware and adjust itself accordingly.

DirectDraw is implemented by the DDRAW dynamic-link library (DLL). This 32-bit DLL implements all of the common functionality required by DirectDraw. It performs all of the necessary thunking between Win32 and the 16-bit portions of the HAL, as well as complete parameter validation. It provides management for off-screen display memory, and performs all of the bookkeeping and semantic logic required for DirectDraw. It is responsible for presenting the Component Object Model (COM) interface to the application, hooking hWnds to provide clip lists, and all other device-independent functionality.

DirectDraw HAL

The DirectDraw HAL is hardware dependent and contains only hardware-specific code. The HAL can be implemented in 16 bits, 32 bits, or, on Windows 95, a combination of the two. The HAL is always 32 bits on Windows NT. The HAL can be an integral part of the display driver or a separate DLL that communicates with the display driver through a private interface defined by the driver's creator.

The DirectDraw HAL is implemented by the chip manufacturer, board producer, or original equipment manufacturer (OEM). The HAL implements only the device-dependent code and performs no emulation. If a function is not performed by the hardware, the HAL should not report it as a hardware capability. The HAL should do no parameter validation. The parameters will be validated by DirectDraw before the HAL is invoked.

DirectDraw Software Emulation

DirectDraw's hardware emulation layer (HEL) presents its capabilities to DirectDraw just like a HAL would. By examining these capabilities during application initialization, you can adjust application parameters to provide optimum performance on a variety of platforms. If a DirectDraw HAL is not present or a requested feature is not provided by the hardware, DirectDraw will emulate the missing functionality.

Types of DirectDraw Objects

The DirectDraw object represents the display device. There can be one DirectDraw object for every logical display device in operation. An application development environment, for instance, might have two monitors, one running the application using DirectDraw and one running the development environment using GDI.

A DirectDrawSurface object represents a linear region of display memory that can be directly accessed and manipulated. These display memory addresses may point to visible frame buffer memory (primary surface) or to non-visible buffers (off-screen or overlay surfaces). These non-visible buffers usually reside in display memory, but can be created in system memory if required by the hardware design or if DirectDraw is doing software emulation. An overlay is a surface that can be made visible without altering the pixels it is obscuring. Overlays and sprites are synonymous. A texture map is a surface that can be wrapped onto a 3D surface.

A DirectDrawPalette object represents either a 16 or a 256 color-indexed palette. Palettes are provided for textures, off-screen surfaces, and overlay surfaces, none of which are required to have the same palette as the primary surface.

The DirectDraw object creates DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper objects. DirectDrawPalette and DirectDrawClipper objects must be attached to the DirectDrawSurface objects they affect. A DirectDrawSurface may refuse the request to attach a DirectDrawPalette to it. This is not unusual because most hardware does not support multiple palettes.

Using DirectDraw

IDirectDraw2 Interface

The COM model used by DirectDraw specifies that new functionality can be added by providing new interfaces. This release of DirectDraw implements two new interfaces, the **IDirectDraw2 Interface** and the **IDirectDrawSurface2 Interface**. These new interfaces can be obtained by using the **IDirectDraw::QueryInterface** method, as shown in the following example:

```
/*
 * create an IDirectDraw2 interface
 */
LPDIRECTDRAW        lpDD;
LPDIRECTDRAW2       lpDD2;

ddrval = DirectDrawCreate( NULL, &lpDD, NULL );
if( ddrval != DD_OK )
    return;

ddrval = lpDD->SetCooperativeLevel( hwnd,
         DDSCL_NORMAL );
if( ddrval != DD_OK )
    return;


ddrval = lpDD->QueryInterface( IID_IDirectDraw2,
         (LPVOID *)&lpDD2);
if( ddrval != DD_OK )
    return;


ddscaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
ddrval = lpDD2->GetAvailableVidMem(&ddscaps, &total,
         &free);
if( ddrval != DD_OK )
    return;
```

This example shows C++ syntax for creating an IDirectDraw interface, which then uses the **IDirectDraw::QueryInterface** method to create an IDirectDraw2 interface. This interface contains the **IDirectDraw2::GetAvailableVidMem** method. An attempt to use this method from an IDirectDraw interface will result in a compile-time error.

In addition to the **IDirectDraw2::GetAvailableVidMem** method, the IDirectDraw2 interface contains all of the methods provided in the IDirectDraw interface. Included in this interface are the **IDirectDraw2::SetDisplayMode** and **IDirectDraw2::EnumDisplayModes** methods which allow refresh rates to be specified. If refresh rates are not required, the **IDirectDraw::SetDisplayMode** and **IDirectDraw::EnumDisplayModes** methods can be used.

The interaction between the **IDirectDraw::SetCooperativeLevel** and **IDirectDraw::SetDisplayMode** methods is slightly different than the one between the **IDirectDraw2::SetCooperativeLevel** and **IDirectDraw2::SetDisplayMode** methods. If you are using the IDirectDraw interface and an application gains exclusive mode by calling **IDirectDraw::SetCooperativeLevel** with the DDSCL_EXCLUSIVE flag, changes the mode using **IDirectDraw::SetDisplayMode**, then releases exclusive mode by calling **IDirectDraw::SetCooperativeLevel** with the DDSCL_NORMAL flag, the original display mode will not be restored. The new display mode will remain until the application calls the **IDirectDraw::RestoreDisplayMode** method or the DirectDraw object is deleted. However, if you are using the IDirectDraw2 interface and an application follows the same steps, the original display mode will be restored when exclusive mode is lost.

Note Because some methods behave somewhat differently in the two interfaces, mixing methods from IDirectDraw and IDirectDraw2 may cause unpredictable results. You should only use functions from one of these interfaces at a time; do not use some functions from IDirectDraw and other functions from IDirectDraw2.

For more information on using the IDirectDrawSurface2 interface, see **IDirectDrawSurface2 Interface**.

Multiple DirectDraw Objects per Process

DirectDraw allows a process to call the **DirectDrawCreate** function as many times as necessary. A unique and independent interface will be returned from each call. Each DirectDraw object can be used as desired; there are no dependencies between the objects. Each object behaves exactly as if it had been created by two different processes.

Since the DirectDraw objects are independent, the DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper objects created with a particular DirectDraw object should not be used with other DirectDraw objects because these objects are automatically released when the DirectDraw object is destroyed. If they are used with another DirectDraw object, they may stop functioning if the original object is destroyed.

The exception is DirectDrawClipper objects created with the **DirectDrawCreateClipper** function. These objects are independent of any particular DirectDraw object and can be used with one or more DirectDraw objects.

Support for High Resolutions and True Color Bit Depths

DirectDraw supports all of the screen resolutions and depths supported by the display device driver. DirectDraw allows an application to change the mode to any one supported by the computer's display driver, including all supported 24- and 32-bpp modes.

DirectDraw also supports hardware emulation layer (HEL) blitting of 24- and 32-bpp surfaces. If the display device driver supports blitting at these resolutions, the hardware blitter will be used for display memory to display memory blits. Otherwise, the HEL will be used to perform the blits.

Windows 95 allows a user to specify the type of monitor that is being used. DirectDraw checks the display modes that it knows about against the display restrictions of the installed monitor. If it is determined that the requested mode is not compatible with the monitor, the **IDirectDraw2::SetDisplayMode** method call will fail. Only modes that are supported on the installed monitor will be enumerated with the **IDirectDraw2::EnumDisplayModes** method.

Primary Surface Resource Sharing Model

DirectDraw has a simple resource sharing model. Display memory is a scarce, shared resource. If the mode is changed, all of the surfaces stored in display memory are lost (for more information, see **Losing Surfaces**).

DirectDraw implicitly creates a GDIPrimarySurface when it is instantiated for a display device DirectDraw is sharing with GDI. GDI is granted shared access to the primary surface. DirectDraw keeps track of the surface memory that GDI recognizes as the primary surface. The DirectDrawSurface that owns GDI's primary surface can always be obtained using the **IDirectDraw::GetGDISurface** method.

GDI is not allowed to cache fonts, brushes, and device-dependent bitmaps (DDBs) in the display memory managed by DirectDraw. The HAL must reserve whatever display memory the DIB engine driver needs before describing the available memory to DirectDraw's heap manager or before the display device driver can allocate and free memory for its cached data from DirectDraw's heap manager.

Changing Modes and Exclusive Access

Display modes can be changed using the **IDirectDraw2::SetDisplayMode** method. Modes can be changed by any application as long as all of the applications are sharing the display card.

The DirectDraw exclusive mode does not bar other applications from allocating DirecDrawSurfaces, nor does it exclude them from using DirectDraw or GDI functionality. However, it does prevent all applications, other than the one that obtained exclusive access, from changing the display mode or changing the palette.

Creating DirectDraw Objects Using CoCreateInstance

You can create a DirectDraw object using **CoCreateInstance** and the **IDirectDraw::Initialize** method rather than the **DirectDrawCreate** function. The following steps describe how to create the DirectDraw object:

1 Initialize COM at the start of your application using CoInitialize(NULL).

```
    if (FAILED(CoInitialize(NULL)))
        return FALSE;
```

2 Create your DirectDraw object using **CoCreateInstance** and the **IDirectDraw::Initialize** method.

```
ddrval = CoCreateInstance(&CLSID_DirectDraw,
                                       NULL,
                                       CLSCTX_ALL,
                                       &IID_IDirectDraw,
                                       &lpdd);
if( !FAILED(ddrval) )
    ddrval = IDirectDraw_Initialize( lpdd, NULL);
```

CLSID_DirectDraw is the class identifier of the DirectDraw driver object class and *IID_IDirectDraw* is the particular DirectDraw interface you want. *lpdd* is the DirectDraw object returned. **CoCreateInstance** returns an uninitialized object.

3 Before you use the DirectDraw object, you must call **IDirectDraw::Initialize**. This method takes the driver GUID parameter that the **DirectDrawCreate** function typically uses (NULL in this case). Once the DirectDraw object is initialized, you can use and release the DirectDraw object as if it had been created using the **DirectDrawCreate** function. If you do not call the **IDirectDraw::Initialize** method before using one of the methods associated with the DirectDraw object, a DDERR_NOTINITIALIZED error will occur.

Before closing the application, shut down COM using **CoUninitialize**, as shown below.

```
 CoUnitialize();
```

Using DirectDrawSurface

IDirectDrawSurface2 Interface

The COM model that DirectDraw uses specifies that new functionality can be added by providing new interfaces. This release of DirectDraw implements two new interfaces, the **IDirectDraw2 Interface** and the **IDirectDrawSurface2 Interface**.

The following example shows how to create an **IDirectDrawSurface2** interface:

```
LPDIRECTDRAWSURFACE     lpSurf;
LPDIRECTDRAWSURFACE2    lpSurf2;

// Create surfaces
memset( &ddsd, 0, sizeof( ddsd ) );
ddsd.dwSize = sizeof( ddsd );
ddsd.dwFlags = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN |
                      DDSCAPS_SYSTEMMEMORY;
ddsd.dwWidth = 10;
ddsd.dwHeight = 10;

ddrval = lpDD2->CreateSurface( &ddsd, &lpSurf,
         NULL );
if( ddrval != DD_OK )
    return;

ddrval = lpSurf->QueryInterface(
         IID_IDirectDrawSurface2, (LPVOID *)&lpSurf2);
if( ddrval != DD_OK )
    return;

ddrval = lpSurf2->PageLock( 0 );
if( ddrval != DD_OK )
    return;

ddrval = lpSurf2->PageUnlock( 0 );
if( ddrval != DD_OK )
    return;
```

The IDirectDrawSurface2 interface contains all of the methods provided in the IDirectDrawSurface interface, along with three new methods: **IDirectDrawSurface2::GetDDInterface**, **IDirectDrawSurface2::PageLock**, and **IDirectDrawSurface2::PageUnlock**.

For more information on obtaining the IDirectDraw2 interface, see **IDirectDraw2 Interface**.

Frame Buffer Access

DirectDrawSurface objects represent surface memory in the DirectDraw architecture. A DirectDrawSurface allows an application to directly gain access to this surface memory through the **IDirectDrawSurface::Lock** method. An application calls this method, providing a RECT structure that specifies the rectangle on the surface it requires access to. If the application calls **IDirectDrawSurface::Lock** with a NULL RECT, it is assumed that exclusive access to the entire piece of surface memory is being requested by the application. This method fills in a **DDSURFACEDESC** structure with the information needed for the application to gain access to the surface memory. This information includes the pitch (or stride) and the pixel format of the surface, if different from the pixel format of the primary surface. When an application is finished with the surface memory, the surface memory can be made available with the **IDirectDrawSurface::Unlock** method.

Experience shows that there are several common problems you may encounter when rendering directly into a DirectDrawSurface. The following list describes some solutions to these problems:

- Never assume a constant display pitch. Always examine the pitch information returned by the **IDirectDrawSurface::Lock** method. This pitch may vary for a number of reasons, including the location of the surface memory, the type of display card, or even the version of the DirectDraw driver being used.
- Limit activity between the calls to the **IDirectDrawSurface::Lock** and **IDirectDrawSurface::Unlock** methods. The **IDirectDrawSurface::Lock** method holds the WIN16 lock so that gaining access to surface memory can occur safely, and the **IDirectDrawSurface::GetDC** method implicitly calls **IDirectDrawSurface::Lock**. The WIN16 lock serializes access to GDI and USER, shutting down Windows for the duration between the Lock and Unlock operations, as well as between the GetDC and ReleaseDC operations.
- Be sure you copy aligned to display memory. Windows 95 uses a page fault handler, Vflatd.386, to implement a virtual flat frame buffer for display cards with bank-switched memory. This module allows these display devices to present a linear frame buffer to DirectDraw. Copying unaligned to display memory can cause the system to suspend operations if the copy happens to span memory banks.

Losing Surfaces

The surface memory associated with a DirectDrawSurface object may be freed, while the DirectDrawSurface objects representing these pieces of surface memory are not necessarily released. When a DirectDrawSurface object loses its surface memory, many methods will return **DDERR_SURFACELOST** and perform no other function.

Surfaces can be lost because the display card mode was changed or because an application received exclusive access to the display card and freed all of the surface memory currently allocated on the card. The **IDirectDrawSurface::Restore** method recreates these lost surfaces and reconnects them to their DirectDrawSurface objects.

Surface Format Support in the HEL

The following table shows the pixel formats for off-screen plain surfaces supported by the DirectX 2 HEL. The "Masks" column shows the red, green, blue, and alpha masks for each set of pixel format flags and bit depth.

Pixel Format Flags	**Bit Depth**	**Masks**
DDPF_RGB | DDPF_PALETTEINDEXED1	1	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED4	4	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED8	8	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB	16	R: 0x0000F800 G: 0x000007E0 B: 0x0000001F A: 0x00000000
DDPF_RGB	16	R: 0x00007C00

		G: 0x000003E0
		B: 0x0000001F
		A: 0x00000000
DDPF_RGB	24	R: 0x00FF0000
		G: 0x0000FF00
		B: 0x000000FF
		A: 0x00000000
DDPF_RGB	24	R: 0x000000FF
		G: 0x0000FF00
		B: 0x00FF0000
		A: 0x00000000
DDPF_RGB	32	R: 0x00FF0000
		G: 0x0000FF00
		B: 0x000000FF
		A: 0x00000000
DDPF_RGB	32	R: 0x000000FF
		G: 0x0000FF00
		B: 0x00FF0000
		A: 0x00000000

In addition to supporting a wider range of off-screen surface formats, the HEL also supports surfaces intended for use by Direct3D, or other 3D renderers.

Color and Format Conversion

Non-RGB surface formats are described by FourCC codes. When the pixel format is requested, if the surface is a non-RGB surface, the DDPF_FOURCC flag will be set and the **dwFourCC** member in the **DDPIXELFORMAT** structure will be valid. If the FourCC code represents a YUV format, the DDPF_YUV flag will also be set and the **dwYUVBitCount**, **dwYBits**, **dwUBits**, **dwVBits**, and **dwYUVAlphaBits** members will be valid masks that can be used to extract information from the pixels.

If an RGB format is present, the DDPF_RGB flag will be set and the **dwRGBBitCount**, **dwRBits**, **dwGBits**, **dwBBits**, and **dwRGBAlphaBits** members will be valid masks that can be used to extract information from the pixels. The DDPF_RGB flag can be set in conjunction with the DDPF_FOURCC flag if a non-standard RGB format is being described.

During color and format conversion, two sets of FourCC codes are exposed to the application. One set of FourCC codes represents what the blitting hardware is capable of; the other represents what the overlay hardware is capable of.

Color Keying

Source and destination color keying for blits and overlays are supported by DirectDraw. For both of these types of color keying, a color key or a color range may be supplied.

Source color keying specifies a color or color range that, in the case of blitting, will not be copied, or, in the case of overlays, not be visible on the destination. Destination color keying specifies a color or color range that, in the case of blitting, will be replaced or, in the case of overlays, be covered up on the destination. The source color key specifies what can and can't be read from the source surface. The destination color key specifies what can and can't be written onto, or covered up, on the destination surface. If a destination surface has a color key, only the pixels that match the color key will be changed, or covered up, on the destination surface.

Some hardware will only support color ranges for YUV pixel data. This is because YUV data is usually video, and, due to quantitization errors during conversion, the transparent background is not a single color.

Content should be authored to a single transparent color whenever possible, regardless of pixel format.

Specifying Color Keys

Color keys are specified in the pixel format of a surface. If a surface is in a palettized format, the color key is specified as an index or a range of indices. If the surface's pixel format is specified by a FourCC code that describes a YUV format, the YUV color key is specified by the three low-order bytes in both the **dwColorSpaceLowValue** and **dwColorSpaceHighValue** members of the **DDCOLORKEY** structure. The lowest order byte has the V data, the second lowest order byte has the U data, and the highest order byte has the Y data. The **IDirectDrawSurface::SetColorKey** method has a flags parameter that specifies whether the color key is to be used for overlay operations or blit operations, and whether it is a source or a destination key. Some examples of valid color keys follow:

8-bit palettized mode

```
// palette entry 26 is the color key
dwColorSpaceLowValue = 26;
dwColorSpaceHighValue = 26;
```

24-bit true color mode

```
// color 255,128,128 is the color key
dwColorSpaceLowValue = RGBQUAD(255,128,128);
dwColorSpaceHighValue = RGBQUAD(255,128,128);
```

FourCC YUV mode

```
// any YUV color where Y is between 100 and 110 and U
//or V is between 50 and 55 is transparent
dwColorSpaceLowValue = YUVQUAD(100,50,50);
dwColorSpaceHighValue = YUVQUAD(110,55,55);
```

Flipping Surfaces and GDI's Frame Rate

DirectDraw has extended flipping surfaces to encompass more than page flipping and more than visible surface flipping. Any surface can now be constructed as a flipping surface. This has many advantages over the traditional, limited scope of page flipping.

When an **IDirectDrawSurface::Flip** method operation is requested in DirectDraw, the surface memory areas associated with the DirectDrawSurface objects being flipped are switched. Surfaces attached to the DirectDrawSurface objects being flipped are not affected. For example, in a double-buffered situation, an application that draws on the back buffer always uses the same DirectDrawSurface object. The surface memory underneath the object is just switched with the front buffer when the **IDirectDrawSurface::Flip** method is requested.

If the front buffer is visible, either because it is the primary surface or because it is an overlay that is currently visible, subsequent calls to the **IDirectDrawSurface::Lock** method or the **IDirectDrawSurface::Blt** method that target the back buffer will fail with the **DDERR_WASSTILLDRAWING** return value until the next vertical refresh occurs. This behavior occurs because the front buffer's previous surface memory, which is no longer attached to the back buffer, is still being drawn to the physical display by the hardware. This situation disappears during the next vertical refresh because the hardware that updates the physical display re-reads the location of the display memory on every refresh.

This physical requirement makes calling the **IDirectDrawSurface::Flip** method on visible surfaces an asynchronous command. A good practice to follow when building games, for example, is to perform all of the non-visual elements of the game after this method is called. When the input, audio, game play, and system memory drawing operations have been completed, begin the drawing tasks that require gaining access to the visible back buffers.

When your application needs to run in a window and still requires a flipping environment, it will attempt to create a flipping overlay surface. If the hardware does not support overlays, you can create a primary surface that page flips, and when a surface that GDI is not aware of is about to become the primary surface, blit the contents of the primary surface that GDI is writing to onto the buffer that is about to become visible. This takes little, if any, processing time because the blits are performed asynchronously. It can, however, consume considerable blitter bandwidth that is dependent on screen resolution and the size of the window that is being page flipped. As long as the frame rate does not dip below 20 frames a second, GDI will appear to be operating correctly.

Before you instantiate a DirectDraw object, GDI is already using your display memory to display itself. When you call DirectDraw to instantiate a primary surface, the memory address of that surface will be the same as GDI is currently using.

If you create a complex surface with a back buffer, GDI will first point to the display memory for the primary surface. Since GDI has no knowledge of DirectDraw, GDI will continue operating on this surface, even if you have flipped it and it is now the non-visible back buffer.

Many applications will begin by creating one large window that covers the entire screen. As long as your application is active and has the focus, GDI will not attempt to write into its copy of the buffer since nothing it controls needs redrawing.

For other scenarios, always remember that GDI only knows about the original surface, and never knows if it is currently the primary surface or a back buffer. If you do not need the GDI screen, then use the above technique. If you do need GDI, you can try this technique:

- Create a primary surface with two back buffers.
- Blit the initial primary surface (the GDI surface) to the middle back buffer.
- Flip(NULL) to put GDI into last place and make your initial copy visible.

After you have done this, copy from the GDI buffer to the middle buffer, draw what you want the user to see on that buffer, then use the code below, which keeps GDI safely on the bottom and oscillates between the other two buffers.

```
pPrimary->Flip(pMiddle)
```

Overlay Z-Order

Overlay z-order determines the order in which overlays clip each other, enabling a hardware sprite system to be implemented under DirectDraw. Overlays are assumed to be on top of all other screen components. Destination color keys are only affected by the bits on the primary surface, not by overlays occluded by other overlays. Source color keys work on an overlay whether or not it has a z-order specified. Overlays that do not have a specified z-order behave in unspecified ways when overlaying the same area on the primary surface. Finally, overlays without a specified z-order are assumed to have a z-order of 0. The possible z-order of overlays begins with 0, just on top of the primary surface, and ends with 4 billion, just underneath the glass on the monitor. An overlay with a z-order of 2 would obscure an overlay with a z-order of 1. An overlay is not allowed to have the same z-order as another overlay.

Palettes and Pixel Formats

DirectDraw enables the creation of multiple palettes that can be attached to off-screen surfaces. When this is done, the off-screen surfaces no longer share the palette of the primary surface. If an off-screen surface with a pixel format different from the primary surface is created, it is assumed that the hardware can use it. For instance, if a paletized off-screen surface is created when the primary surface is in 16-bit color mode, it is assumed that the blitter can convert palettized surfaces to true color during the blit operation.

DirectDraw supports the creation of standard 8-bit palettized surfaces, capable of displaying 256 colors, and two kinds of 4-bit palettized surfaces, each capable of displaying 16 colors. The first kind of 4-bit palettized surface is indexed into a true color palette table; the second kind is indexed into the primary surface indexed palette table. This second type of palette provides 50 percent compression and a layer of indirection to the sprites stored using it.

If these surfaces are to be created, the blitter must be able replace the palette during the blit operation. Therefore, while blitting from one palettized surface to another, the palette is ignored. Palette decoding is only done to true color surfaces, or when the 4-bit palette is an index to an index in the 8-bit palette. In all other cases, the indexed palette is the palette of the destination.

Raster operations for palettized surfaces are ignored. Changing the attached palette of a surface is a very quick operation. All three of these palettized surfaces should be supported as textures on 3D accelerated hardware.

Blitting To and From System Memory Surfaces

Some display cards have DMA hardware that allows them to efficiently blit to and from system memory surfaces. The **DDCAPS** structure has been expanded to allow drivers to report this capability. The following members have been added:

```
DWORD   dwSVBCaps
DWORD   dwSVBCKeyCaps
DWORD   dwSVBFXCaps
DWORD   dwSVBRops[DD_ROP_SPACE]

DWORD   dwVSBCaps
DWORD   dwVSBCKeyCaps
DWORD   dwVSBFXCaps
DWORD   dwVSBRops[DD_ROP_SPACE]

DWORD   dwSSBCaps
DWORD   dwSSBCKeyCaps
DWORD   dwSSBFXCaps
DWORD   dwSSBRops[DD_ROP_SPACE]
```

The SVB prefix indicates capabilities values that relate system memory to display memory blits. The VSB prefix indicates capabilities values that relate display memory to system memory blits. The SSB prefix indicates capabilities values that relate system memory to system memory blits.

The **dwSVBCaps** member corresponds to the **dwCaps** member except that it describes the blitting capabilities of the display driver for system memory to display memory blits. Likewise, the **dwSVBCKeyCaps** member corresponds to the **dwCKeyCaps** member and the **dwSVBFXCaps** member corresponds to the **dwFXCaps** member. The **dwSVBRops** member array describes the raster operations the driver supports for this type of blit.

These members are only valid if the **DDCAPS_CANBLTSYSMEM** flag is set in **dwCaps**, indicating that the driver is able to blit to or from system memory.

If the system memory surface being used by the hardware blitter is not locked, DirectDraw will automatically call the **IDirectDrawSurface2::PageLock** method on the surface to ensure that the memory has been locked.

Using DirectDrawPalette

Setting Palettes on Non-Primary Surfaces

In DirectX 2, palettes can be attached to any palettized surface (primary, back buffer, off-screen plain, or texture map). Only those palettes attached to primary surfaces will have any effect on the system palette. It is important to note that DirectDraw blits never perform color conversion—any palettes attached to the source or destination surface of a blit are ignored. Furthermore, the DirectDraw surface method, **IDirectDrawSurface::GetDC**, also ignores any DirectDrawPalette selected into the surface.

Non-primary surface palettes are intended for use by applications or Direct3D, or other 3D renderers.

Sharing Palettes

In DirectX 2, palettes can be shared between multiple surfaces. The same palette can be set on the front and back buffers of a flipping chain or shared between multiple texture surfaces. When a palette is attached to a surface with the **IDirectDrawSurface::SetPalette** method, the surface increments the reference count of that palette. When the reference count of the surface reaches 0, it will decrement the reference count of the attached palette. In addition, if a palette is detached from a surface by calling **IDirectDrawSurface::SetPalette** with a NULL palette interface pointer, the reference count of the surface's palette will be decremented.

Note If the **IDirectDrawSurface::SetPalette** method is called several times consecutively for the same surface with the same palette, the reference count for the palette will be incremented only once. Subsequent calls will not affect the palette's reference count.

New Palette Types

In DirectX 2, DirectDraw supports 1-bit (2 entry), 2-bit (4 entry) and 4-bit (16 entry) palettes in addition to the 8-bit (256 entry) palettes supported by previous versions. Such palettes can be created by specifying one of the new palette capability flags: **DDPCAPS_1BIT**, **DDPCAPS_2BIT**, and **DDPCAPS_4BIT**. Matching capability flags have been added for surface pixel formats: **DDPF_PALETTEINDEXED1**, **DDPF_PALETTEINDEXED2**, and **DDPF_PALETTEINDEXED4**.

A palette can only be attached to a surface with a matching pixel format. For example, a 2 entry palette created with the DDPCAPS_1BIT flag can only be attached to a 1-bit surface created with the pixel format flag DDPF_PALETTEINDEXED1.

Furthermore, it is now possible to create indexed palettes. An indexed palette is one whose entries do not hold RGB colors, but rather integer indices into the array of PALETTEENTRYs of some target palette. An indexed palette's color table is an array of 2, 4, 16, or 256 bytes, where each byte is an index into some unspecified, destination palette.

To create an indexed palette, specify the palette capability flag DDPCAPS_8BITENTRIES when calling the **IDirectDraw::CreatePalette** method. For example, to create a 4-bit, indexed palette, specify DDPCAPS_4BIT | DDPCAPS_8BITENTRIES. When creating an indexed palette, a pointer to an array of bytes is passed rather than a pointer to an array of PALETTEENTRY structures. The pointer to the array of bytes must be cast to an LPPALETTEENTRY when calling **IDirectDraw::CreatePalette**.

Using DirectDrawClipper

Driver Independent Clippers

You can create clipper objects that are not directly owned by any particular DirectDraw object. Such clipper objects can be shared across multiple

DirectDraw objects. Driver independent clipper objects are created with the new DirectDraw API function **DirectDrawCreateClipper**. This function can be called before any DirectDraw objects are created.

Because these clippers are not owned by any DirectDraw object, they are not automatically released when your application's objects are released. If not released explicitly by the application, these clippers will be released by DirectDraw when the application terminates.

You can still create clippers with the **IDirectDraw::CreateClipper** method. These DirectDrawClipper objects are automatically released when the DirectDraw object from which they were created is released.

Clip Lists

Clip lists are managed by DirectDraw using the DirectDrawClipper object. A DirectDrawClipper can be attached to any surface. A window handle can also be attached to a DirectDrawClipper, in which case DirectDraw will update the DirectDrawClipper clip list with the clip list from the window as it changes.

Although the clip list is visible from the DirectDraw HAL, DirectDraw will only call the HAL for blitting with rectangles that meet the clip list requirements. For instance, if the upper right rectangle of a surface was clipped and the application directed DirectDraw to blit the surface onto the primary surface, DirectDraw would have the HAL do two blits. The first blit would be the upper left hand corner of the surface and the second would be the bottom half of the surface.

The HAL considers the clip list for overlays only if the overlay hardware can support clipping and if destination color keying is not active. Most of today's hardware does not support occluded overlays unless they are being destination color keyed. This can be reported to DirectDraw as a driver capability, in which case the overlay will be turned off if it becomes occluded. Under these conditions, the HAL does not consider clip lists either.

Sharing Clippers

In DirectX 2, clippers can be shared between multiple surfaces. For example, the same clipper can be set on both the front and back buffers of a flipping chain. When a clipper is attached to a surface by using the **IDirectDrawSurface::SetClipper** method, the surface increments the reference count of that clipper. When the reference count of the surface reaches 0, it will decrement the reference count of the attached clipper. In addition, if a clipper is detached from a surface by calling **IDirectDrawSurface::SetClipper** with a NULL clipper interface pointer, the reference count of the surface's clipper will be decremented.

Note If **IDirectDrawSurface::SetClipper** is called several times consecutively on the same surface for the same clipper, the reference count for the clipper will be incremented only once. Subsequent calls will not affect the clipper's reference count.

Creating Clipper Objects Using CoCreateInstance

DirectDrawClipper objects have full class-factory support for COM compliance. In addition to the standard **DirectDrawCreateClipper** function and **IDirectDraw::CreateClipper** method, you can also create a DirectDrawClipper object either by using **CoGetClassObject** to obtain a class factory and then calling **CoCreateInstance**, or by calling **CoCreateInstance** directly. The following example shows how to create a DirectDrawClipper object using **CoCreateInstance** and the **IDirectDrawClipper::Initialize** method.

```
ddrval = CoCreateInstance(&CLSID_DirectDrawClipper,
                                      NULL,
                                      CLSCTX_ALL,
                                      &IID_IDirectDrawClipper,
                                      &lpClipper);
if (!FAILED(ddrval))
    ddrval = IDirectDrawClipper_Initialize(lpClipper,
       lpDD, 0UL);
```

CLSID_DirectDrawClipper is the class identifier of the DirectDrawClipper object class, *IID_IDirectDrawClipper* is the currently supported interface (which is the one you want), and *lpClipper* is the clipper object returned.

Clippers created by the class factory mechanism must be initialized with the **IDirectDrawClipper::Initialize** method before you can use the object. 0UL is the *dwFlags* parameter, which in this case has a value of 0 since no flags are currently supported. In the example shown here, *lpDD* is the DirectDraw object that owns the DirectDrawClipper object. However, you could supply a NULL instead, which would create an independent clipper (equivalent to creating a DirectDrawClipper object using the **DirectDrawCreateClipper** function).

Before closing the application, shut down COM using **CoUninitialize**, as shown below.

```
    CoUnitialize();
```

Support for 3D Surfaces

Texture Maps

In DirectX 2, texture maps can be allocated in system memory using the HEL. To allocate a texture map surface, specify the **DDSCAPS_TEXTURE** flag in the **ddsCaps** member of the surface description passed to the **IDirectDraw::CreateSurface** method.

A wide range of texture pixel formats are supported by the HEL. The following table describes these formats. The "Masks" column contains the red, green, blue, and alpha masks for each set of pixel format flags and bit depths.

Pixel Format Flags	**Bit Depth**	**Masks**
DDPF_RGB | DDPF_PALETTEINDEXED1	1	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED1 | DDPF_PALETTEINDEXEDTO8	1	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2 | DDPF_PALETTEINDEXEDTO8	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED4	4	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB |	4	R: 0x00000000

DDPF_PALETTEINDEXED4 | DDPF_PALETTEINDEXEDTO8		G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED8	8	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB	8	R: 0x000000E0 G: 0x0000001C B: 0x00000003 A: 0x00000000
DDPF_RGB | DDPF_ALPHAPIXELS	16	R: 0x00000F00 G: 0x000000F0 B: 0x0000000F A: 0x0000F000
DDPF_RGB	16	R: 0x0000F800 G: 0x000007E0 B: 0x0000001F A: 0x00000000
DDPF_RGB	16	R: 0x0000001F G: 0x000007E0 B: 0x0000F800 A: 0x00000000
DDPF_RGB	16	R: 0x00007C00 G: 0x000003E0 B: 0x0000001F A: 0x00000000
DDPF_RGB | DDPF_ALPHAPIXELS	16	R: 0x00007C00 G: 0x000003E0

		B: 0x0000001F A: 0x00008000
DDPF_RGB	24	R: 0x00FF0000 G: 0x0000FF00 B: 0x000000FF A: 0x00000000
DDPF_RGB	24	R: 0x000000FF G: 0x0000FF00 B: 0x00FF0000 A: 0x00000000
DDPF_RGB	32	R: 0x00FF0000 G: 0x0000FF00 B: 0x000000FF A: 0x00000000
DDPF_RGB	32	R: 0x000000FF G: 0x0000FF00 B: 0x00FF0000 A: 0x00000000
DDPF_RGB | DDPF_ALPHAPIXELS	32	R: 0x00FF0000 G: 0x0000FF00 B: 0x000000FF A: 0xFF000000
DDPF_RGB | DDPF_ALPHAPIXELS	32	R: 0x000000FF G: 0x0000FF00 B: 0x00FF0000 A: 0xFF000000

The formats shown in the previous table are those that can be created by the HEL in system memory. The DirectDraw device driver for a 3D-accelerated display card is free to create textures of other formats in display memory. Such a driver should export the DDSCAPS_TEXTURE flag to indicate that it can create textures, and should be prepared to handle the DirectDraw HAL callback

CanCreateSurface to verify that the surface description for a texture map is one the driver is prepared to create.

Mipmaps

In DirectX 2, DirectDraw supports mipmapped texture surfaces. A mipmap is a sequence of textures, each of which is a progressively lower resolution, prefiltered representation of the same image. Mipmapping is a computationally low-cost way of improving the quality of rendered textures. Each pre-filtered image, or level, in the mipmap is a power of two smaller than the previous level. In DirectDraw, mipmaps are represented as a chain of attached surfaces. The highest resolution texture is at the head of the chain and has, as an attachment, the next level of the mipmap which has, in turn, an attachment that is the next level in the mipmap, and so on down to the lowest resolution level of the mipmap.

To create a surface representing a single level of a mipmap, specify the **DDSCAPS_MIPMAP** flag in the surface description passed to the **IDirectDraw::CreateSurface** method. Because all mipmaps are also textures, the **DDSCAPS_TEXTURE** flag must also be specified. It is possible to create each level manually and build the chain with the **IDirectDrawSurface::AddAttachedSurface** method. However, the **IDirectDraw::CreateSurface** method can be used to build an entire mipmap chain in a single operation.

The following example demonstrates building a chain of five mipmap levels of sizes 256×256, 128×128, 64×64, 32×32 and 16×16.

```
DDSURFACEDESC       ddsd;
LPDIRECTDRAWSURFACE lpDDMipMap;
ZeroMemory(&ddsd, sizeof(ddsd));
ddsd.dwSize = sizeof(ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_MIPMAPCOUNT;
ddsd.dwMipMapCount = 5;
ddsd.ddsCaps.dwCaps = DDSCAPS_TEXTURE |
        DDSCAPS_MIPMAP | DDSCAPS_COMPLEX;
ddsd.dwWidth = 256UL;
ddsd.dwHeight = 256UL;

ddres = lpDD->CreateSurface(&ddsd, &lpDDMipMap);
if (FAILED(ddres))
        ...
```

You can omit the number of mipmaps levels, in which case the **IDirectDraw::CreateSurface** method will create a chain of surfaces, each a power of two smaller than the previous one, down to the smallest possible size. It is also possible to omit the width and height, in which case

IDirectDraw::CreateSurface will create the number of levels you specify with a minimum level size of 1×1.

A chain of mipmap surfaces is traversed using the **IDirectDrawSurface::GetAttachedSurface** method, specifying the DDSCAPS_MIPMAP and DDSCAPS_TEXTURE capability flags. The following example traverses a mipmap chain from highest to lowest resolutions.

```
LPDIRECTDRAWSURFACE lpDDLevel, lpDDNextLevel;
DDSCAPS ddsCaps;

lpDDLevel = lpDDMipMap;
lpDDLevel->AddRef();
ddsCaps.dwCaps = DDSCAPS_TEXTURE | DDSCAPS_MIPMAP;
ddres = DD_OK;
while (ddres == DD_OK)
{
        // Process this level
        ...
        ddres = lpDDLevel->GetAttachedSurface(
                &ddsCaps, &lpDDNextLevel);
        lpDDLevel->Release();
        lpDDLevel = lpDDNextLevel;
}
if ((ddres != DD_OK) && (ddres != DDERR_NOTFOUND))
        ...
```

You can also build flippable chains of mipmaps. In this scenario, each mipmap level has an associated chain of back buffer texture surfaces. Each back buffer texture is attached to one level of the mipmap. Only the front buffer in the chain has the **DDSCAPS_MIPMAP** flag set; the others are simply texture maps (DDSCAPS_TEXTURE). A mipmap level can have two attached texture maps, one with DDSCAPS_MIPMAP set, which is the next level in the mipmap chain, and one with the **DDSCAPS_BACKBUFFER** flag set, which is the back buffer of the flippable chain. All the surfaces in each flippable chain must be of the same size.

It is not possible to build such a surface arrangement with a single call to the **IDirectDraw::CreateSurface** method. To construct a flippable mipmap, either build a complex mipmap chain and manually attach back buffers with the **IDirectDrawSurface::AddAttachedSurface** method or create a sequence of flippable chains and build the mipmap with **IDirectDrawSurface::AddAttachedSurface**.

Note Blit operations apply to only a single level in the mipmap chain. To blit an entire chain of mipmaps, each level must be blitted separately.

The **IDirectDrawSurface::Flip** method will flip all the levels of a mipmap from the level supplied to the lowest level in the map. A destination surface can also be provided, in which case all levels in the mipmap will flip to the back buffer in their flippable chain. This back buffer matches the supplied override. For example, if the third back buffer in the top-level flippable chain is supplied, all levels in the mipmap will flip to their third back buffer.

The number of levels in a mipmap chain is stored explicitly. When the surface description of a mipmap is obtained (using **IDirectDrawSurface::Lock** or **IDirectDrawSurface::GetSurfaceDesc**), the **dwMipMapCount** member will contain the number of levels in a mipmap, including the top level. For levels other than the top level in the map, **dwMipMapCount** will specify the number of levels from that map to the smallest map in the chain.

Z-Buffers

In DirectX 2, the DirectDraw HEL can create z-buffers for use by Direct3D or other 3D rendering software. The HEL supports both 16- and 32-bit z-buffers. The DirectDraw device driver for a 3D-accelerated display card can permit the creation of z-buffers in display memory by exporting the surface capability DDSCAPS_ZBUFFER. It should also specify the z-buffer depths it supports using the **dwZBufferBitDepths** member of the **DDCAPS** structure.

Z-buffers can be cleared using the **IDirectDrawSurface::Blt** method. A new DirectDraw blit flag (DDBLT_DEPTHFILL) has been defined to indicate that the blit clears z-buffers. If this flag is specified, the **DDBLTFX** structure passed to the **IDirectDrawSurface::Blt** method should have its **dwFillDepth** member set to the required z-depth. If the DirectDraw device driver for a 3D-accelerated display card is designed to provide support for z-buffer clearing in hardware, it should export the capability flag DDCAPS_BLTDEPTHFILL and should have code to handle DDBLT_DEPTHFILL blits. The destination surface of a depth fill blit must be a z-buffer.

Note The actual interpretation of a depth value is 3D-renderer specific.

Direct3D Integration with DirectDraw

Direct3D Driver Interface

DirectDraw presents a single, unified object to you as an application programmer. This object encapsulates both the DirectDraw and Direct3D states. The DirectDraw driver COM interfaces (IID_IDirectDraw or IID_IDirectDraw2) and the Direct3D driver COM interface (IID_IDirect3D) all allow you to communicate with the same underlying object. Therefore, no Direct3D object is created. Rather, a Direct3D interface to the DirectDraw object is obtained. This is achieved using the standard COM **QueryInterface** method.

The following example demonstrates how to create the DirectDraw object and obtain a Direct3D interface for communicating with that object.

```
LPDIRECTDRAW lpDD;
LPDIRECT3D   lpD3D;
ddres = DirectDrawCreate(NULL, &lpDD, NULL);
if (FAILED(ddres))
        ...
ddres = lpDD->QueryInterface(IID_IDirect3D,
        &lpD3D);
if (FAILED(ddres))
        ...
```

The code shown in the previous example creates a single object and obtains two interfaces to that object. Therefore, the object's reference count after the **IDirectDraw::QueryInterface** method call is two. The important implication of this is that the lifetime of the Direct3D driver state is the same as that of the DirectDraw object. Releasing the Direct3D interface does not destroy the Direct3D driver state. That state is not destroyed until all references, whether DirectDraw or Direct3D, to that object have been released. Hence, if you release a Direct3D interface while holding a reference to a DirectDraw driver interface, and re-query the Direct3D interface, the Direct3D state will be preserved.

Direct3D Device Interface

As with the object, there is no distinct Direct3D device object. A Direct3D device is simply an interface for communicating with a DirectDraw surface used as a 3D rendering target. The following example creates a Direct3D device interface to a DirectDrawSurface object.

```
LPDIRECTDRAWSURFACE lpDDSurface;
LPDIRECT3DDEVICE    lpD3DDevice;

ddres = lpDD->CreateSurface(&ddsd, &lpDDSurface,
        NULL);
if (FAILED(ddres))
        ...
ddres = lpDDSurface->QueryInterface(lpGuid,
        &lpD3DDevice);
if (FAILED(ddres))
        ...
```

The same rules for reference counts and state lifetimes for objects (see **Direct3D Driver Interface**) apply to DirectDraw surfaces and Direct3D devices. Additionally, multiple, distinct Direct3D device interfaces can be obtained for the same DirectDraw surface. It is possible, therefore, that a single DirectDraw surface could be the target for both a ramp-based device and an RGB-based device.

Direct3D Texture Interface

Direct3D textures are not distinct object types, but rather another interface of DirectDrawSurface objects. The following example obtains a Direct3D texture interface from a DirectDrawSurface object.

```
LPDIRECTDRAWSURFACE lpDDSurface;
LPDIRECT3DTEXTURE   lpD3DTexture;

ddres = lpDD->CreateSurface(&ddsd, &lpDDSurface,
        NULL);
if (FAILED(ddres))
        ...
ddres = lpDDSurface->QueryInterface(
         IID_IDirect3DTexture, &lpD3DTexture);
if (FAILED(ddres))
        ...
```

The same rules for reference counts and state lifetimes discussed for objects (see **Direct3D Driver Interface**) apply to Direct3D textures. It is possible to use a single DirectDrawSurface as both a rendering target and a texture.

DirectDraw HEL and Direct3D

The DirectDraw HEL has been enhanced to support the creation of texture, mipmap, and z-buffer surfaces. Furthermore, due to the tight integration of DirectDraw and Direct3D, a DirectDraw-enabled system will always provide Direct3D support (in software emulation, at least). Hence, the DirectDraw HEL exports the DDSCAPS_3DDEVICE flag to indicate that a surface can be used for 3D rendering. A DirectDraw driver for a hardware-accelerated 3D display card should export this capability to indicate the presence of hardware-accelerated 3D.

Reference

Functions

Two of the DirectDraw functions initiate control of display memory through the IDirectDraw interface. The first function, **DirectDrawCreate**, creates an instance of a DirectDraw object that represents a specific piece of display hardware. The second function, **DirectDrawEnumerate**, obtains a list of all DirectDraw objects installed on a system. These functions are the mechanisms DirectDraw uses to support multiple pieces of display hardware. To support multiple display devices, your application need only select a specific DirectDraw object and instantiate it.

The third function, **DirectDrawCreateClipper**, creates an instance of a DirectDraw clipper that is not directly owned by a DirectDraw object, and can be shared across multiple driver objects.

DirectDrawCreate

```
HRESULT DirectDrawCreate(GUID FAR * lpGUID,
    LPDIRECTDRAW FAR * lplpDD,IUnknown FAR * pUnkOuter);
```

Creates an instance of a DirectDraw object.

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_DIRECTDRAWALREADYCREATED

  DDERR_GENERIC

  DDERR_INVALIDDIRECTDRAWGUID

  DDERR_INVALIDPARAMS

  DDERR_NODIRECTDRAWHW

  DDERR_OUTOFMEMORY

lpGUID
: Address of the GUID that represents the driver to be created. NULL is always the active display driver.

lplpDD
: Address of a pointer that will be initialized with a valid DirectDraw pointer if the call succeeds.

pUnkOuter
: Allows for future compatibility with COM aggregation features. Presently, however, **DirectDrawCreate** will return an error if this parameter is anything but NULL.

This function attempts to initialize a DirectDraw object, then sets a pointer to the object if successful. Calling the **IDirectDraw::GetCaps** method immediately after initialization is advised to determine to what extent this object is hardware accelerated.

DirectDrawCreateClipper

```
HRESULT DirectDrawCreateClipper( DWORD dwFlags,
    LPDIRECTDRAWCLIPPER FAR *lplpDDClipper,
    IUnknown FAR *pUnkOuter);
```

Creates an instance of a DirectDrawClipper object not associated with a DirectDraw object.

To create a DirectDrawClipper object owned by a specific DirectDraw object, use the **IDirectDraw::CreateClipper** method.

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_INVALIDPARAMS **DDERR_OUTOFMEMORY**

dwFlags
: This parameter is not currently used and must be set to 0.

lplpDDClipper
: Address of a pointer to be filled in with the address of the new DirectDrawClipper object.

pUnkOuter
: Allows for future compatibility with COM aggregation features. Presently, however, **DirectDrawCreateClipper** will return an error if this parameter is anything but NULL.

This function can be called before any DirectDraw objects are created. Because these clippers are not owned by any DirectDraw object, they are not automatically released when an application's objects are released. If not released explicitly by the application, such clippers will be released by DirectDraw when the application terminates.

See also **IDirectDraw::CreateClipper**

DirectDrawEnumerate

```
HRESULT DirectDrawEnumerate(LPDDENUMCALLBACK lpCallback,
    LPVOID lpContext);
```

Enumerates the DirectDraw objects installed on the system. The NULL GUID entry always identifies the primary display device shared with GDI.

- Returns DD_OK if successful, or **DDERR_INVALIDPARAMS** otherwise.

lpCallback
: Address of a **Callback** function that will be called with a description of each DirectDraw-enabled HAL installed in the system.

lpContext
: Address of a caller-defined context that will be passed to the enumeration callback each time it is called.

Callback Functions

Most of the functionality of DirectDraw is provided by the methods of its Component Object Model (COM) interfaces. This section lists the callback functions that are not implemented as part of a COM interface.

Callback

```
BOOL WINAPI lpCallback(GUID FAR * lpGUID,
    LPSTR lpDriverDescription, LPSTR lpDriverName,
    LPVOID lpContext);
```

Application-defined callback procedure for the **DirectDrawEnumerate** function.

- Returns DDENUMRET_OK to continue the enumeration, or DDENUMRET_CANCEL to stop it.

lpGUID
Address of the unique identifier of the DirectDraw object.

lpDriverDescription
Address of a string containing the driver description.

lpDriverName
Address of a string containing the driver name.

lpContext
Address of a caller-defined structure that will be passed to the callback function each time the function is invoked.

EnumCallback

```
HRESULT WINAPI lpEnumCallback(LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSURFACEDESC lpDDSurfaceDesc,LPVOID lpContext);
```

Application-defined callback procedure for the **IDirectDraw::EnumSurfaces** method.

- Returns DDENUMRET_OK to continue the enumeration, or DDENUMRET_CANCEL to stop it.

lpDDSurface
Address of the DirectDrawSurface currently being enumerated if it is an *existing* surface (DDENUMSURFACES_DOESEXIST). The value will be NULL if a *potential* surface is being enumerated (DDENUMSURFACES_CANBECREATED).

lpDDSurfaceDesc
Address of the **DDSURFACEDESC** structure for the existing or potential surface that most closely matches the requested surface.

lpContext
Address of the caller-defined structure passed to the member every time it is invoked.

EnumModesCallback

```
HRESULT WINAPI lpEnumModesCallback(LPDDSURFACEDESC lpDDSurfaceDesc,
    LPVOID lpContext);
```

Application-defined callback procedure for the **IDirectDraw2::EnumDisplayModes** method.

- Returns DDENUMRET_OK to continue the enumeration, or DDENUMRET_CANCEL to stop it.

lpDDSurfaceDesc
Address of the **DDSURFACEDESC** structure that provides the monitor frequency and the mode that can be created. This data is read-only.

lpContext
Address of a caller-defined structure that will be passed to the callback function each time the function is invoked.

EnumSurfacesCallback

```
HRESULT WINAPI lpEnumSurfacesCallback(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSURFACEDESC lpDDSurfaceDesc,LPVOID lpContext);
```

Application-defined callback procedure for the **IDirectDrawSurface::EnumAttachedSurfaces** method.

- Returns DDENUMRET_OK to continue the enumeration, or DDENUMRET_CANCEL to stop it.

lpDDSurface
Address of the surface attached to this surface.

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure that describes the attached surface.

lpContext
Address of the user-defined context specified by the user.

fnCallback

```
HRESULT WINAPI lpfnCallback(LPDIRECTDRAWSURFACE lpDDSurface,
    LPVOID lpContext);
```

Application-defined callback procedure for the **IDirectDrawSurface::EnumOverlayZOrders** method.

- Returns DDENUMRET_OK to continue the enumeration, or DDENUMRET_CANCEL to stop it.

lpDDSurface
Address of the surface being overlaid on this surface.

lpContext
Address of the user-defined context specified by the user.

IDirectDraw Interface

DirectDraw objects represent the display hardware. An object is hardware-accelerated if the display device for which it was instantiated has hardware acceleration. Three types of objects can be created by a DirectDraw object: DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper.

More than one DirectDraw object can be instantiated at a time. The simplest example of this would be using two monitors on a Windows 95 system. Although Windows 95 does not support dual monitors natively, it is possible to write a DirectDraw HAL for each display device. The display device Windows 95 and GDI recognizes is the one that will be used when the default DirectDraw object is instantiated. The display device that Windows 95 and GDI does not recognize can be addressed by another, independent DirectDraw object that must be created using the second display device's identifying GUID. This GUID can be obtained through the **DirectDrawEnumerate** function.

The DirectDraw object manages all of the objects it creates. It controls the default palette if the primary surface is in 8 bpp mode, the default color key values, and the hardware display mode. It tracks what resources have been allocated and what resources remain to be allocated.

Changing the display mode is an important piece of DirectDraw functionality. The display mode resolution can be changed at any time unless another application has obtained exclusive access to DirectDraw. The pixel depth of the display mode can only be changed if the application requesting the change has obtained exclusive access to the DirectDraw object. All DirectDrawSurface objects lose surface memory and become inoperative when the mode is changed. A surface's memory must be reallocated using the **IDirectDrawSurface::Restore** method.

IDirectDraw Interface Method Groups

Applications use the methods of the IDirectDraw interface to create DirectDraw objects and work with system-level variables. The methods can be organized into the following groups:

Allocating memory	**Compact**
	Initialize
Creating objects	**CreateClipper**

	CreatePalette
	CreateSurface
Device capabilities	**GetCaps**
Display modes	**EnumDisplayModes**
	GetDisplayMode
	GetMonitorFrequency
	RestoreDisplayMode
	SetDisplayMode
Display status	**GetScanLine**
	GetVerticalBlankStatus
IUnknown	**AddRef**
	QueryInterface
	Release
Miscellaneous	**GetAvailableVidMem**
	GetFourCCCodes
	WaitForVerticalBlank
Setting behavior	**SetCooperativeLevel**
Surfaces	**DuplicateSurface**
	EnumSurfaces
	FlipToGDISurface
	GetGDISurface

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the DirectDraw object without affecting the functionality of the original interface.

IDirectDraw::AddRef

```
ULONG AddRef();
```

Increases the reference count of the DirectDraw object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

When the DirectDraw object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirectDraw::Release** method to decrease the reference count of the object by 1.

See also **IDirectDraw::Initialize**, **IDirectDraw::QueryInterface**, **IDirectDraw::Release**

IDirectDraw::Compact

```
HRESULT Compact();
```

At present this method is only a stub; it has not yet been implemented.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_SURFACEBUSY	**DDERR_NOEXCLUSIVEMODE**

This method moves all of the pieces of surface memory on the display card to a contiguous block to make the largest single amount of free memory available. This call will fail if any operations are in progress.

The application calling this method must have its cooperative level set to exclusive.

IDirectDraw::CreateClipper

```
HRESULT CreateClipper(DWORD dwFlags,
    LPDIRECTDRAWCLIPPER FAR * lplpDDClipper,
    IUnknown FAR * pUnkOuter);
```

Creates a DirectDrawClipper object.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_OUTOFMEMORY	**DDERR_NOCOOPERATIVELEVELSET**

dwFlags
: This parameter is not currently used and must be set to 0.

lplpDDClipper
Address of a pointer to be filled in with the address of the new DirectDrawClipper object if the **IDirectDraw::CreateClipper** method is successful.

pUnkOuter
Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw::CreateClipper** will return an error if this parameter is anything but NULL.

The DirectDrawClipper object can be attached to a DirectDrawSurface and used during **IDirectDrawSurface::Blt**, **IDirectDrawSurface::BltBatch**, and **IDirectDrawSurface::UpdateOverlay** operations.

To create a DirectDrawClipper object that is not owned by a specific DirectDraw object, use the **DirectDrawCreateClipper** function.

See also **IDirectDrawSurface::GetClipper**, **IDirectDrawSurface::SetClipper**

IDirectDraw::CreatePalette

```
HRESULT CreatePalette(DWORD dwFlags,
    LPPALETTEENTRY lpColorTable,
    LPDIRECTDRAWPALETTE FAR * lplpDDPalette,
    IUnknown FAR * pUnkOuter);
```

Creates a DirectDrawPalette object for this DirectDraw object.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_NOEXCLUSIVEMODE**
DDERR_INVALIDPARAMS	**DDERR_NOCOOPERATIVELEVELSET**
DDERR_OUTOFCAPS	**DDERR_OUTOFMEMORY**
DDERR_UNSUPPORTED	

dwFlags

DDPCAPS_1BIT
Indicates the index is 1 bit. There are two entries in the palette table.

DDPCAPS_2BIT
Indicates the index is 2 bits. There are four entries in the palette table.

DDPCAPS_4BIT
Indicates the index is 4 bits. There are sixteen entries in the palette table.

DDPCAPS_8BITENTRIES
An index to an 8-bit color index. This flag is only valid when used with the DDPCAPS_1BIT, DDPCAPS_2BIT, or DDPCAPS_4BIT flag, and when the target surface is in 8-bpp. Each color entry is one byte long and is an index to a destination surface's 8-bpp palette.

DDPCAPS_8BIT

Indicates the index is 8 bits. There are 256 entries in the palette table.

DDPCAPS_ALLOW256

Indicates this palette can have all 256 entries defined.

lpColorTable
Address of an array of 2, 4, 16, or 256 PALETTEENTRY structures that will initialize this DirectDrawPalette object.

lplpDDPalette
Address of a pointer to be filled in with the address of the new DirectDrawPalette object if the **IDirectDraw::CreatePalette** method is successful.

pUnkOuter
Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw::CreatePalette** returns an error if this parameter is anything but NULL.

IDirectDraw::CreateSurface

```
HRESULT CreateSurface(LPDDSURFACEDESC lpDDSurfaceDesc,
    LPDIRECTDRAWSURFACE FAR * lplpDDSurface,
    IUnknown FAR * pUnkOuter);
```

Creates a DirectDrawSurface object for this DirectDraw object.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INCOMPATIBLEPRIMARY

DDERR_INVALIDCAPS

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_INVALIDPIXELFORMAT

DDERR_NOALPHAHW

DDERR_NOCOOPERATIVELEVELSET

DDERR_NODIRECTDRAWHW

DDERR_NOEMULATION

DDERR_NOEXCLUSIVEMODE

DDERR_NOFLIPHW

DDERR_NOMIPMAPHW

DDERR_NOZBUFFERHW

DDERR_OUTOFMEMORY

DDERR_OUTOFVIDEOMEMORY

DDERR_PRIMARYSURFACEALREADYEXISTS

DDERR_UNSUPPORTEDMODE

lpSurfaceDesc
Address of the **DDSURFACEDESC** structure that describes the requested surface.

lplpDDSurface
Address of a pointer to be initialized with a valid DirectDrawSurface pointer if the call succeeds.

pUnkOuter
Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw::CreateSurface** will return an error if this parameter is anything but NULL.

The DirectDrawSurface object represents a surface (pixel memory) that usually resides in the display card memory, but can exist in system memory if display memory is exhausted or if it is explicitly requested. If the hardware cannot support the capabilities requested or if it previously allocated those resources to another DirectDrawSurface object, the call to **IDirectDraw::CreateSurface** will fail.

This method usually creates one DirectDrawSurface object. If the DDSCAPS_FLIP flag in the **dwCaps** member of the **DDSCAPS** structure is set, **IDirectDraw::CreateSurface** will create several DirectDrawSurface objects, referred to collectively as a *complex structure*. The additional surfaces created are also referred to as *implicit* surfaces.

DirectDraw does not permit the creation of display memory surfaces wider than the primary surface.

The following are examples of valid surface creation scenarios:

Scenario 1

The primary surface is the surface currently visible to the user. When you create a primary surface, you are actually creating a DirectDrawSurface object to access an already existing surface being used by GDI. Consequently, while all other types of surfaces require *dwHeight* and *dwWidth* values, a primary surface must not have them specified, even if you know they are the same dimensions as the existing surface.

The members of the **DDSURFACEDESC** structure (*ddsd* below) relevant to the creation of the primary surface are then filled in.

```
DDSURFACEDESC    ddsd;
ddsd.dwSize = sizeof( ddsd );

//Tell DDRAW which fields are valid
ddsd.dwFlags = DDSD_CAPS;
```

```
//Ask for a primary surface
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;
```

Scenario 2

Create a simple off-screen surface of the type that might be used to cache bitmaps that will later be composed with the blitter. A height and width are required for all surfaces except primary surfaces. The members in the **DDSURFACEDESC** structure (*ddsd* below) relevant to the creation of a simple off-screen surface are then filled in.

```
DDSURFACEDESC    ddsd;
ddsd.dwSize = sizeof( ddsd );

//Tell DDRAW which fields are valid
ddsd.dwFlags = DDSD_CAPS | DDSD_HEIGHT | DDSD_WIDTH;

//Ask for a simple off-screen surface, sized
//100 by 100 pixels
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
dwHeight = 100;
dwWidth = 100;
```

DirectDraw creates this surface in display memory unless it will not fit, in which case the surface is created in system memory. If the surface must be created in one or the other, use the flags DDSCAPS_SYSTEMMEMORY or DDSCAPS_VIDEOMEMORY in **dwCaps** to specify system memory or display memory, respectively. An error is returned if the surface cannot be created in the specified location.

DirectDraw also allows for the creation of complex surfaces. A complex surface is a set of surfaces created with a single call to the **IDirectDraw::CreateSurface** method. If the DDSCAPS_COMPLEX flag is set in the **IDirectDraw::CreateSurface** call, one or more *implicit* surfaces will be created by DirectDraw in addition to the surface explicitly specified. Complex surfaces are managed as a single surface—a single call to the **IDirectDraw::Release** method will release all surfaces in the structure, and a single call to the **IDirectDrawSurface::Restore** method will restore them all.

Scenario 3

One of the most useful complex surfaces you can specify is composed of a primary surface and one or more back buffers that form a surface flipping environment. The members in the **DDSURFACEDESC** structure (*ddsd* below) relevant to complex surface creation are filled in to describe a flipping surface that has one back buffer.

```
DDSURFACEDESC      ddsd;
```

```
    ddsd.dwSize = sizeof( ddsd );

    //Tell DDRAW which fields are valid
    ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;

    //Ask for a primary surface with a single
    //back buffer
    ddsd.ddsCaps.dwCaps = DDSCAPS_COMPLEX | DDSCAPS_FLIP |
    DDSCAPS_PRIMARYSURFACE;
    ddsd.dwBackBufferCount = 1;
```

The previous statements construct a double-buffered flipping environment—a single call to the **IDirectDrawSurface::Flip** method exchanges the surface memory of the primary surface and the back buffer. If a *BackBufferCount* of 2 is specified, two back buffers are created, and each call to **IDirectDrawSurface::Flip** rotates the surfaces in a circular pattern, providing a triple-buffered flipping environment.

IDirectDraw::DuplicateSurface

```
HRESULT DuplicateSurface(LPDIRECTDRAWSURFACE lpDDSurface,
    LPLPDIRECTDRAWSURFACE FAR * lplpDupDDSurface);
```

Duplicates a DirectDrawSurface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_CANTDUPLICATE	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_OUTOFMEMORY**
DDERR_SURFACELOST	

lpDDSurface
Address of the DirectDrawSurface structure to be duplicated.

lplpDupDDSurface
Address of the DirectDrawSurface pointer that points to the newly created duplicate DirectDrawSurface structure.

This method creates a new DirectDrawSurface object that points to the same surface memory as an existing DirectDrawSurface object. This duplicate can be used just like the original object. The surface memory is released after the last object referencing it is released. A primary surface, 3D surface, or implicitly created surface cannot be duplicated.

IDirectDraw2::EnumDisplayModes

```
HRESULT EnumDisplayModes(DWORD dwFlags,
    LPDDSURFACEDESC lpDDSurfaceDesc, LPVOID lpContext,
    LPDDENUMMODESCALLBACK lpEnumModesCallback);
```

Enumerates all of the display modes the hardware exposes through the DirectDraw object that are compatible with a provided surface description. If NULL is passed for the surface description, all exposed modes will be enumerated.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

dwFlags

DDEDM_REFRESHRATES

Enumerate modes with different refresh rates. **IDirectDraw2::EnumDisplayModes** guarantees that a particular mode will be enumerated only once. This flag specifies whether the refresh rate is taken into account when determining if a mode is unique.

DDSD_REFRESHRATE

The **dwRefreshRate** member of the **DDSURFACEDESC** structure is valid.

lpDDSurfaceDesc

Address of a **DDSURFACEDESC** structure that will be checked against available modes. If the value of this parameter is NULL, all modes will be enumerated.

lpContext

Address of a caller-defined structure that will be passed to each enumeration member.

lpEnumModesCallback

Address of the **EnumModesCallback** function the enumeration procedure will call every time a match is found.

This method enumerates the **dwRefreshRate** member in the **DDSURFACEDESC** structure; the **IDirectDraw2::EnumDisplayModes** method does not contain this capability. If you use the **IDirectDraw2::SetDisplayMode** method to set the refresh rate of a new mode, you must use **IDirectDraw2::EnumDisplayModes** to enumerate the **dwRefreshRate** member.

To ensure COM compliance, this method is not part of the IDirectDraw interface, but belongs to the IDirectDraw2 interface. To use this method, you must first query for the IDirectDraw2 interface. For more information, see **IDirectDraw2 Interface**.

See also **IDirectDraw::GetDisplayMode**, **IDirectDraw2::SetDisplayMode**, **IDirectDraw::RestoreDisplayMode**

IDirectDraw::EnumSurfaces

```
HRESULT EnumSurfaces(DWORD dwFlags,
    LPDDSURFACEDESC lpDDSD, LPVOID lpContext,
    LPDDENUMSURFACESCALLBACk lpEnumCallback);
```

Enumerates all of the existing or possible surfaces that meet the search criterion specified.

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_INVALIDOBJECT **DDERR_INVALIDPARAMS**

dwFlags

DDENUMSURFACES_ALL

Enumerates all of the surfaces that meet the search criterion.

DDENUMSURFACES_MATCH

Searches for any surface that matches the surface description.

DDENUMSURFACES_NOMATCH

Searches for any surface that does not match the surface description.

DDENUMSURFACES_CANBECREATED

Enumerates the first surface that can be created that meets the search criterion.

DDENUMSURFACES_DOESEXIST

Enumerates the already existing surfaces that meet the search criterion.

lplpDDSD

Address of a **DDSURFACEDESC** structure that defines the surface of interest.

lpContext

Address of a caller-defined structure to be passed to each enumeration member.

lpEnumCallback

Address of the **EnumCallback** function the enumeration procedure will call every time a match is found.

If the DDENUMSURFACES_CANBECREATED flag is set, this method will attempt to temporarily create a surface that meets the criteria. Note that as a surface is enumerated, its reference count is increased—if you are not going to use the surface, use **IDirectDraw::Release** to release the surface after each enumeration.

IDirectDraw::FlipToGDISurface

```
HRESULT FlipToGDISurface();
```

Makes the surface that GDI writes to the primary surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOTFOUND	

This method can be called at the end of a page flipping application to ensure that the display memory that GDI is writing to is visible to the user.

See also **IDirectDraw::GetGDISurface**

IDirectDraw2::GetAvailableVidMem

```
HRESULT GetAvailableVidMem(LPDDSCAPS lpDDSCaps,
    LPDWORD lpdwTotal, LPDWORD lpdwFree);
```

Gets the total amount of display memory available and the amount of display memory currently free. If NULL is passed to either *lpdwTotal* or *lpdwFree*, the value for that parameter is not returned.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_NODIRECTDRAWHW**
DDERR_INVALIDPARAMS	**DDERR_INVALIDCAPS**

lpDDSCaps
Address of a **DDSCAPS** structure that contains the hardware capabilities of the surface.

lpdwTotal
Address of a doubleword to be filled in with the total amount of display memory available.

lpdwFree
Address of a doubleword to be filled in with the amount of display memory currently free.

The following C++ example demonstrates using **IDirectDraw2::GetAvailableVidMem** to determine both the total and free display memory available for texture map surfaces:

```
LPDIRECTDRAW2 lpDD2;
DDSCAPS       ddsCaps;
DWORD         dwTotal;
DWORD         dwFree;

ddres = lpDD->QueryInterface(IID_IDirectDraw2,
&lpDD2); if (FAILED(ddres))
                  ...
ddsCaps.dwCaps = DDSCAPS_TEXTURE;
```

```
ddres = lpDD2->GetAvailableVidMem(&ddsCaps,
&dwTotal, &dwFree);
if (FAILED(ddres))
        ...
```

This method only gives a snapshot of the current display memory state. The amount of free display memory is subject to change as surfaces are created and released. Therefore, the free memory value should only be used as a rough guide. In addition, a particular display adapter card may make no distinction between two different memory types. For example, it may use the same portion of display memory to store z-buffers and textures. Hence, allocating one type of surface (for example, a z-buffer) may affect the amount of display memory available for another type of surface (for example, textures). Therefore, it is best to first allocate an application's fixed resources (such as front, back and z-buffers) before determining how much memory is available for dynamic use (such as texture mapping).

To ensure COM compliance, this method is not a member of the IDirectDraw interface, but is part of the IDirectDraw2 interface. To use this method, you must first query for the IDirectDraw2 interface. For more information, see **IDirectDraw2 Interface**.

IDirectDraw::GetCaps

```
HRESULT GetCaps(LPDDCAPS lpDDDriverCaps,
    LPDDCAPS lpDDHELCaps);
```

Fills in the raw, not remaining, capabilities of the device driver for the hardware and the hardware emulation layer (HEL).

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_INVALIDOBJECT　　**DDERR_INVALIDPARAMS**

lpDDDriverCaps
  Address of a **DDCAPS** structure to be filled in with the capabilities of the hardware, as reported by the device driver.

lpDDHELCaps
  Address of a **DDCAPS** structure to be filled in with the capabilities of the HEL.

See also **DDCAPS**

IDirectDraw::GetDisplayMode

```
HRESULT GetDisplayMode(LPDDSURFACEDESC lpDDSurfaceDesc);
```

Returns the current display mode.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTEDMODE	

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure to be filled in with a description of the surface.

An application should not save the information returned by **IDirectDraw::GetDisplayMode** to restore the display mode on clean up. The mode restoration on clean up should be performed with **IDirectDraw::RestoreDisplayMode**, thereby avoiding mode setting conflicts that could arise in a multiprocess environment.

See also **IDirectDraw2::SetDisplayMode**, **IDirectDraw::RestoreDisplayMode**, **IDirectDraw2::EnumDisplayModes**

IDirectDraw::GetFourCCCodes

```
HRESULT GetFourCCCodes(LPDWORD lpNumCodes,
    LPDWORD lpCodes);
```

Returns the FourCCCodes supported by the DirectDraw object. **IDirectDraw::GetFourCCCodes** can also return the number of codes supported.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lpNumCodes
Address of a doubleword that contains the number of entries the *lpCodes* array can hold. If the number of entries is too small to accommodate all the codes, *lpNumCodes* will be set to the required number and the *lpCodes* array will be filled with all that will fit.

lpCodes
Address of an array of doublewords to be filled in with FourCC codes supported by this DirectDraw object. If NULL is passed, *lpNumCodes* will be set to the number of supported FourCC codes and the method will return.

IDirectDraw::GetGDISurface

```
HRESULT GetGDISurface(
    LPDIRECTDRAWSURFACE FAR * lplpGDIDDSSurface);
```

Returns the DirectDrawSurface object that currently represents the surface memory GDI is treating as the primary surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOTFOUND	

lplpGDIDDSSurface
Address of a DirectDrawSurface pointer to the DirectDrawSurface object that currently controls GDI's primary surface memory.

See also **IDirectDraw::FlipToGDISurface**

IDirectDraw::GetMonitorFrequency

```
HRESULT GetMonitorFrequency(LPDWORD lpdwFrequency);
```

Returns the frequency of the monitor being driven by the DirectDraw object. The frequency value is returned in Hz multiplied by 100. For example, 60Hz is returned as 6000.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTED	

lpdwFrequency
Address of the doubleword to be filled in with the monitor frequency.

IDirectDraw::GetScanLine

```
HRESULT GetScanLine(LPDWORD lpdwScanLine);
```

Returns the scan line that is currently being drawn on the monitor.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTED	**DDERR_VERTICALBLANKINPROGRESS**

lpdwScanLine
Address of the doubleword to contain the scan line the display is currently on.

See also **IDirectDraw::GetVerticalBlankStatus**, **IDirectDraw::WaitForVerticalBlank**

IDirectDraw::GetVerticalBlankStatus

```
HRESULT GetVerticalBlankStatus(LPBOOL lpbIsInVB);
```

Returns the status of the vertical blank.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lpbIsInVB
Address of the BOOL to be filled in with the status of the vertical blank.

This method will set the passed BOOL to TRUE if a vertical blank is occurring and to FALSE otherwise. To synchronize with the vertical blank, use the **IDirectDraw::WaitForVerticalBlank** method.

See also **IDirectDraw::GetScanLine**, **IDirectDraw::WaitForVerticalBlank**

IDirectDraw::Initialize

```
HRESULT Initialize(GUID FAR * lpGUID);
```

Initializes the DirectDraw object.

- Returns **DDERR_ALREADYINITIALIZED**.

lpGUID
Address of the GUID used as the interface identifier.

This method is provided for compliance with the Component Object Model (COM) protocol. Since the DirectDraw object is initialized when it is created, calling this method will always result in the DDERR_ALREADYINITIALIZED return value.

See also **IDirectDraw::AddRef**, **IDirectDraw::QueryInterface**, **IDirectDraw::Release**

IDirectDraw::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID FAR * ppvObj);
```

Determines if the DirectDraw object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

riid
Reference identifier of the interface being requested.

ppvObj
: Address of a pointer to be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirectDraw::QueryInterface** method allows DirectDraw objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

See also **IDirectDraw::AddRef**, **IDirectDraw::Initialize**, **IDirectDraw::Release**

IDirectDraw::Release

```
ULONG Release();
```

Decreases the reference count of the DirectDraw object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

The DirectDraw object deallocates itself when its reference count reaches 0. Use the **IDirectDraw::AddRef** method to increase the reference count of the object by 1.

See also **IDirectDraw::AddRef**, **IDirectDraw::QueryInterface**, **IDirectDraw::Initialize**

IDirectDraw::RestoreDisplayMode

```
HRESULT RestoreDisplayMode();
```

Resets the mode of the display device hardware for the primary surface to what it was before the **IDirectDraw2::SetDisplayMode** method was called to change it. Exclusive level access is required to use this method.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_LOCKEDSURFACES**
DDERR_NOEXCLUSIVEMODE	

See also **IDirectDraw2::SetDisplayMode**, **IDirectDraw::RestoreDisplayMode**, **IDirectDraw2::EnumDisplayModes**, **IDirectDraw::SetCooperativeLevel**

IDirectDraw::SetCooperativeLevel

```
HRESULT SetCooperativeLevel(HWND hWnd, DWORD dwFlags);
```

Determines the top-level behavior of the application.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_HWNDALREADYSET	**DDERR_INVALIDPARAMS**
DDERR_HWNDSUBCLASSED	**DDERR_INVALIDOBJECT**
DDERR_EXCLUSIVEMODEALREADYSET	**DDERR_OUTOFMEMORY**

hWnd
Specifies the window handle used for the application.

dwFlags

DDSCL_ALLOWMODEX
Allows the use of ModeX display modes.

DDSCL_ALLOWREBOOT
Allows CTRL_ALT_DEL to function while in full screen exclusive mode.

DDSCL_EXCLUSIVE
Requests the exclusive level.

DDSCL_FULLSCREEN
Indicates that the exclusive mode owner will be responsible for the entire primary surface. GDI can be ignored.

DDSCL_NORMAL
Indicates that the application will function as a regular Windows application.

DDSCL_NOWINDOWCHANGES
Indicates that DirectDraw is not allowed to minimize or restore the application window on activation.

The DDSCL_EXCLUSIVE flag must be set to call functions that can have drastic performance consequences for other applications. To call the **IDirectDraw::Compact** method, change the display mode, or modify the behavior (for example, flipping) of the primary surface, an application must be set to the exclusive level. If an application calls the **IDirectDraw::SetCooperativeLevel** method with DDSCL_EXCLUSIVE and DDSCL_FULLSCREEN flags set, DirectDraw will attempt to resize its window to full screen. An application must either set the DDSCL_EXCLUSIVE or DDSCL_NORMAL flags, and DDSCL_EXCLUSIVE requires DDSCL_FULLSCREEN.

ModeX modes are only available if an application sets DDSCL_ALLOWMODEX, DDSCL_FULLSCREEN, and DDSCL_EXCLUSIVE. DDSCL_ALLOWMODEX cannot be used with DDSCL_NORMAL. If DDSCL_ALLOWMODEX is not specified, the **IDirectDraw2::EnumDisplayModes** method will not enumerate the ModeX modes, and the **IDirectDraw2::SetDisplayMode** method will fail when a ModeX mode is requested. The set of supported display modes may change after using **IDirectDraw::SetCooperativeLevel**.

ModeX modes are not supported by Windows; therefore, when in a ModeX mode you cannot use the **IDirectDrawSurface::Lock** method to lock the primary surface, or the **IDirectDrawSurface::Blt** method to blit to the primary surface. Use the **IDirectDrawSurface::GetDC** method on the primary surface, or use GDI with a screen DC. ModeX modes are indicated by the DDSCAPS_MODEX flag in the **DDSCAPS** member of the **DDSURFACEDESC** structure returned by the **IDirectDrawSurface::GetCaps** and **IDirectDraw2::EnumDisplayModes** methods.

Because applications can use DirectDraw with multiple windows, **IDirectDraw::SetCooperativeLevel** does not require an HWND to be specified if the application is requesting the DDSCL_NORMAL mode. By passing a NULL to HWND, all of the windows can be used simultaneously in normal Windows mode.

See also **IDirectDraw2::SetDisplayMode**, **IDirectDraw::Compact**, **IDirectDraw2::EnumDisplayModes**

IDirectDraw2::SetDisplayMode

```
HRESULT SetDisplayMode(DWORD dwWidth, DWORD dwHeight,
    DWORD dwBPP, DWORD dwRefreshRate, DWORD dwFlags);
```

Sets the mode of the display device hardware.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDMODE**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_LOCKEDSURFACES	**DDERR_NOEXCLUSIVEMODE**
DDERR_SURFACEBUSY	**DDERR_UNSUPPORTED**
DDERR_UNSUPPORTEDMODE	**DDERR_WASSTILLDRAWING**

dwWidth
Specifies the width of the new mode.

dwHeight
Specifies the height of the new mode.

dwBPP
Specifies the bits per pixel of the new mode.

dwRefreshRate
Specifies the refresh rate of the new mode. If this parameter is set to 0, the IDirectDraw interface version of this method is used.

dwFlags
This parameter is not currently used and must be set to 0.

The **IDirectDraw::SetCooperativeLevel** method must be used to set exclusive level access before the mode can be changed. If other applications have created a DirectDrawSurface object on the primary surface and the mode is changed, those applications' primary surface objects will return DDERR_SURFACELOST until they are restored.

To ensure COM compliance, this method is not part of the IDirectDraw interface, but belongs to the IDirectDraw2 interface. To use this method, you must first query for the IDirectDraw2 interface. For more information, see **IDirectDraw2 Interface**.

See also **IDirectDraw::RestoreDisplayMode**, **IDirectDraw::GetDisplayMode**, **IDirectDraw2::EnumDisplayModes**, **IDirectDraw::SetCooperativeLevel**

IDirectDraw::WaitForVerticalBlank

```
HRESULT WaitForVerticalBlank(DWORD dwFlags,
    HANDLE hEvent);
```

Helps the caller synchronize itself with the vertical blank interval.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTED	**DDERR_WASSTILLDRAWING**

dwFlags
Determines how long to wait for the vertical blank.

DDWAITVB_BLOCKBEGIN
Returns when the vertical blank interval begins.

DDWAITVB_BLOCKBEGINEVENT
Triggers an event when the vertical blank begins. This is not currently supported.

DDWAITVB_BLOCKEND
Returns when the vertical blank interval ends and the display begins.

hEvent
Handle for the event to be triggered when the vertical blank begins.

See also **IDirectDraw::GetVerticalBlankStatus**, **IDirectDraw::GetScanLine**

IDirectDrawSurface Interface

The DirectDrawSurface object represents a two-dimensional piece of memory that contains data. This data is in a form understood by the display hardware represented by the DirectDraw object that created the DirectDrawSurface object. A DirectDrawSurface object is created by the **IDirectDraw::CreateSurface** method. Although it is not required, the DirectDrawSurface object usually resides in the display RAM of the display card. Unless specifically stated during DirectDrawSurface object creation, the DirectDraw object will put the DirectDrawSurface object wherever the best performance can be achieved given the requested capabilities.

DirectDrawSurface objects can take advantage of specialized processors on display cards, not only to perform certain tasks faster, but to perform some tasks in parallel with the system central processing unit.

DirectDrawSurface objects recognize, and are integrated with, the rest of the components of the Windows display system. DirectDrawSurface objects can create handles to Window GDI device contexts (HDCs) that allow GDI functions to write to the surface memory represented by the DirectDrawSurface object. GDI perceives these HDCs as memory device contexts, but the hardware accelerators are usually enabled for them if they are in display memory.

IDirectDrawSurface Interface Method Groups

Applications use the methods of the IDirectDrawSurface interface to create DirectDrawSurface objects and work with system-level variables. The methods can be organized into the following groups:

Allocating memory	**Initialize**
	IsLost
	Restore
Attaching surfaces	**AddAttachedSurface**
	DeleteAttachedSurface
	EnumAttachedSurfaces
	GetAttachedSurface
Blitting	**Blt**
	BltBatch
	BltFast

Color keys	**GetColorKey**
	SetColorKey
Device contexts	**GetDC**
	ReleaseDC
Flipping surfaces	**Flip**
IUnknown	**AddRef**
	QueryInterface
	Release
Locking surfaces	**Lock**
	PageLock
	PageUnlock
	Unlock
Miscellaneous	**GetDDInterface**
Overlays	**AddOverlayDirtyRect**
	EnumOverlayZOrders
	GetOverlayPosition
	SetOverlayPosition
	UpdateOverlay
	UpdateOverlayDisplay
	UpdateOverlayZOrder
Status	**GetBltStatus**
	GetFlipStatus
Surface capabilities	**GetCaps**
Surface clipper	**GetClipper**
	SetClipper
Surface description	**GetPixelFormat**

	GetSurfaceDesc
Surface palettes	**GetPalette**
	SetPalette

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the DirectDrawSurface object without affecting the functionality of the original interface.

IDirectDrawSurface::AddAttachedSurface

```
HRESULT AddAttachedSurface(
    LPDIRECTDRAWSURFACE lpDDSAttachedSurface);
```

Attaches a surface to another surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_CANNOTATTACHSURFACE

  DDERR_GENERIC

  DDERR_INVALIDOBJECT

  DDERR_INVALIDPARAMS

  DDERR_SURFACEALREADYATTACHED

  DDERR_SURFACELOST

  DDERR_WASSTILLDRAWING

lpDDSAttachedSurface
Address of the DirectDrawSurface that is to be attached.

Possible attachments include z-buffers, alpha channels, and back buffers. Some attachments automatically break other attachments. For example, the 3DZBUFFER can only be attached to one back buffer at a time. Attachment is not bi-directional, and a surface cannot be attached to itself. Emulated surfaces (in system memory) cannot be attached to non-emulated surfaces. Unless one surface is a texture map, the two attached surfaces must be the same size. A flippable surface cannot be attached to another flippable surface of the same type; however, attaching two surfaces of different types is allowed. For example, a flippable z-buffer can be attached to a regular flippable surface. If a non-flippable surface is attached to another non-flippable surface of the same type, the two surfaces will become a flippable chain. If a non-flippable surface is attached to a flippable surface, it becomes part of the existing flippable chain. Additional surfaces can be added to this chain, and each call of the **IDirectDrawSurface::Flip** method will advance one step through the surfaces.

See also **IDirectDrawSurface::DeleteAttachedSurface**, **IDirectDrawSurface::EnumAttachedSurfaces**, **IDirectDrawSurface::Flip**

IDirectDrawSurface::AddOverlayDirtyRect

```
HRESULT AddOverlayDirtyRect(LPRECT lpRect);
```

Builds the list of the rectangles that need to be updated the next time the **IDirectDrawSurface::UpdateOverlayDisplay** method is called.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTED	**DDERR_INVALIDSURFACETYPE**

lpRect
Address of the RECT structure that needs to be updated.

This method is used for the software implementation. It is not needed if the overlay support is provided by the hardware.

See also **IDirectDrawSurface::UpdateOverlayDisplay**

IDirectDrawSurface::AddRef

```
ULONG  AddRef();
```

Increases the reference count of the DirectDrawSurface object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

When the DirectDraw object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirectDrawSurface::Release** method to decrease the reference count of the object by 1.

See also **IDirectDraw::CreateSurface**, **IDirectDrawSurface::Initialize**, **IDirectDrawSurface::QueryInterface**, **IDirectDrawSurface::Release**

IDirectDrawSurface::Blt

```
HRESULT Blt(LPRECT lpDestRect,
    LPDIRECTDRAWSURFACE lpDDSrcSurface,LPRECT lpSrcRect,
    DWORD dwFlags, LPDDBLTFX lpDDBltFx);
```

Performs a bit block transfer.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDCLIPLIST**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDRECT	**DDERR_NOALPHAHW**
DDERR_NOBLTHW	**DDERR_NOCLIPLIST**
DDERR_NODDROPSHW	**DDERR_NOMIRRORHW**
DDERR_NORASTEROPHW	**DDERR_NOROTATIONHW**
DDERR_NOSTRETCHHW	**DDERR_NOZBUFFERHW**
DDERR_SURFACEBUSY	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

lpDestRect
Address of a RECT structure that defines the upper left and lower right points of the rectangle on the destination surface to be blitted to.

lpDDSrcSurface
Address of the DirectDrawSurface structure that represents the DirectDrawSurface. This is the source for the blit operation.

lpSrcRect
Address of a RECT structure that defines the upper left and lower right points of the rectangle on the source surface to be blitted from.

dwFlags

DDBLT_ALPHADEST

Uses either the alpha information in pixel format or the alpha channel surface attached to the destination surface as the alpha channel for this blit.

DDBLT_ALPHADESTCONSTOVERRIDE

Uses the **dwAlphaDestConst** member in the **DDBLTFX** structure as the alpha channel for the destination surface for this blit.

DDBLT_ALPHADESTNEG

The NEG suffix indicates that the destination surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHADESTSURFACEOVERRIDE

Uses the **lpDDSAlphaDest** member in the **DDBLTFX** structure as the alpha channel for the destination for this blit.

DDBLT_ALPHAEDGEBLEND

Uses the **dwAlphaEdgeBlend** member in the **DDBLTFX** structure as the alpha channel for the edges of the image that border the color key colors.

DDBLT_ALPHASRC

Uses either the alpha information in pixel format or the alpha channel surface attached to the source surface as the alpha channel for this blit.

DDBLT_ALPHASRCCONSTOVERRIDE

Uses the **dwAlphaSrcConst** member in the **DDBLTFX** structure as the alpha channel for the source for this blit.

DDBLT_ALPHASRCNEG

The NEG suffix indicates that the source surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHASRCSURFACEOVERRIDE

Uses the **lpDDSAlphaSrc** member in the **DDBLTFX** structure as the alpha channel for the source for this blit.

DDBLT_ASYNC

Performs this blit asynchronously through the FIFO in the order received. If no room is available in the FIFO hardware, fail the call.

DDBLT_COLORFILL

Uses the **dwFillColor** member in the **DDBLTFX** structure as the RGB color that fills the destination rectangle on the destination surface.

DDBLT_DDFX

Uses the **dwDDFX** member in the **DDBLTFX** structure to specify the effects to use for the blit.

DDBLT_DDROPS

Uses the **dwDDROPS** member in the **DDBLTFX** structure to specify the raster operations that are not part of the Win32 API.

DDBLT_DEPTHFILL

Uses the **dwFillDepth** member in the **DDBLTFX** structure as the depth value with which to fill the destination rectangle on the destination z-buffer surface.

DDBLT_KEYDEST

Uses the color key associated with the destination surface.

DDBLT_KEYDESTOVERRIDE

Uses the **dckDestColorkey** member in the **DDBLTFX** structure as the color key for the destination surface.

DDBLT_KEYSRC

Uses the color key associated with the source surface.

DDBLT_KEYSRCOVERRIDE

Uses the **dckSrcColorkey** member in the **DDBLTFX** structure as the color key for the source surface.

DDBLT_ROP

Uses the **dwROP** member in the **DDBLTFX** structure for the raster operation for this blit. These ROPs are the same as those defined in the Win32 API.

DDBLT_ROTATIONANGLE

Uses the **dwRotationAngle** member in the **DDBLTFX** structure as the angle (specified in 1/100th of a degree) to rotate the surface.

DDBLT_WAIT

Postpones the DDERR_WASSTILLDRAWING return value if the blitter is busy. Returns as soon as the blit can be set up or another error occurs.

DDBLT_ZBUFFER

Performs a z-buffered blit using the z-buffers attached to the source and destination surfaces and the **dwZBufferOpCode** member in the **DDBLTFX** structure as the z-buffer opcode.

DDBLT_ZBUFFERDESTCONSTOVERRIDE

Performs a z-buffered blit using the **dwZDestConst** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the destination.

DDBLT_ZBUFFERDESTOVERRIDE

Performs a z-buffered blit using the **lpDDSZBuffeDestr** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the destination.

DDBLT_ZBUFFERSRCCONSTOVERRIDE

Performs a z-buffered blit using the **dwZSrcConst** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the source.

DDBLT_ZBUFFERSRCOVERRIDE

Performs a z-buffered blit using the **lpDDSZBufferSrc** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the source.

lpDDBltFx

See the **DDBLTFX** structure.

This method is capable of synchronous or asynchronous blits, either display memory to display memory, display memory to system memory, system memory to display memory, or system memory to system memory. The blits can be performed using z-information, alpha information, source color keys and destination color keys. Arbitrary stretching or shrinking will be performed if the source and destination rectangles are not the same size.

Typically, **IDirectDrawSurface::Blt** returns immediately with an error if the blitter is busy and the blit could not be set up. The DDBLT_WAIT flag can alter this behavior so that the method will either wait until the blit can be set up or another error occurs before it returns.

IDirectDrawSurface::BltBatch

```
HRESULT BltBatch(LPDDBLTBATCH lpDDBltBatch,
    DWORD dwCount, DWORD dwFlags);
```

Performs a sequence of **IDirectDrawSurface::Blt** operations from several sources to a single destination. At present this method is only a stub; it has not yet been implemented.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDCLIPLIST**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDRECT	**DDERR_NOALPHAHW**
DDERR_NOBLTHW	**DDERR_NOCLIPLIST**
DDERR_NODDROPSHW	**DDERR_NOMIRRORHW**
DDERR_NORASTEROPHW	**DDERR_NOROTATIONHW**
DDERR_NOSTRETCHHW	**DDERR_NOZBUFFERHW**
DDERR_SURFACEBUSY	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

lpDDBltBatch
Address of the first **DDBLTBATCH** structure that defines the parameters for the blit operations.

dwCount
The number of blit operations to be performed.

dwFlags
This parameter is not used at this time and must be set to 0.

IDirectDrawSurface::BltFast

```
HRESULT BltFast(DWORD dwX, DWORD dwY,
    LPDIRECTDRAWSURFACE lpDDSrcSurface,
    LPRECT lpSrcRect, DWORD dwTrans);
```

Performs a source copy blit or transparent blit using a source or destination color key. This method always attempts an asynchronous blit if this is supported by the hardware.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_EXCEPTION	**DDERR_GENERIC**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDRECT	**DDERR_NOBLTHW**
DDERR_SURFACEBUSY	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

dwX
X-coordinate to blit to on the destination surface.

dwY
Y-coordinate to blit to on the destination surface.

lpDDSrcSurface
Address of the DirectDrawSurface structure that represents the DirectDrawSurface. This is the source for the blit operation.

lpSrcRect
Address of a RECT structure that defines the upper left and lower right points of the rectangle on the source surface to be blitted from.

dwTrans
Specifies the type of transfer.

DDBLTFAST_DESTCOLORKEY
A transparent blit that uses the destination's color key.

DDBLTFAST_NOCOLORKEY
A normal copy blit with no transparency.

DDBLTFAST_SRCCOLORKEY
A transparent blit that uses the source's color key.

DDBLTFAST_WAIT
Postpones the DDERR_WASSTILLDRAWING message if the blitter is busy. Returns as soon as the blit can be set up or another error occurs.

This method only works on display memory surfaces and cannot clip when blitting. The software implementation of **IDirectDrawSurface::BltFast** is 10 percent faster than the **IDirectDrawSurface::Blt** method. However, there is no speed difference between the two if display hardware is being used.

Typically, **IDirectDrawSurface::BltFast** returns immediately with an error if the blitter is busy and the blit cannot be set up. The DDBLTFAST_WAIT flag can be used to alter this behavior so that this method will not return until either the blit can be set up or another error occurs.

IDirectDrawSurface::DeleteAttachedSurface

```
HRESULT DeleteAttachedSurface(DWORD dwFlags,
    LPDIRECTDRAWSURFACE lpDDSAttachedSurface);
```

Detaches two attached surfaces. The detached surface is not released.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_CANNOTDETACHSURFACE**
DDERR_INVALIDPARAMS	**DDERR_SURFACENOTATTACHED**

DDERR_SURFACELOST

dwFlags
This parameter is not used at this time and must be set to 0.

lpDDSAttachedSurface
Address of the DirectDrawSurface structure to be detached. If NULL is passed, all attached surfaces will be detached.

If NULL is passed as the surface to be detached, all attached surfaces will be detached. Implicit attachments, those formed by DirectDraw rather than the **IDirectDrawSurface::AddAttachedSurface** method, cannot be detached. Detaching surfaces from a flippable chain can alter other surfaces in the chain. If a front buffer is detached from a flippable chain, the next surface in the chain becomes the front buffer, and the following surface becomes the back buffer. If a back buffer is detached from a chain, the following surface becomes a back buffer. If a plain surface is detached from a chain, the chain simply becomes shorter. If a flippable chain only has two surfaces and they are detached, the chain is destroyed and both surfaces return to their previous designations.

See also **IDirectDrawSurface::Flip**

IDirectDrawSurface::EnumAttachedSurfaces

```
HRESULT EnumAttachedSurfaces(LPVOID lpContext,
    LPDDENUMSURFACESCALLBACK lpEnumSurfacesCallback );
```

Enumerates all the surfaces attached to a given surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT **DDERR_INVALIDPARAMS**
DDERR_SURFACELOST

lpContext
Address of the caller-defined structure that is passed to the enumeration member every time it is called.

lpEnumSurfacesCallback
Address of the **EnumSurfacesCallback** function that will be called for each surface that is attached to this surface.

IDirectDrawSurface::EnumOverlayZOrders

```
HRESULT EnumOverlayZOrders(DWORD dwFlags,
    LPVOID lpContext,
    LPDDENUMSURFACESCALLBACK lpfnCallback);
```

Enumerates the overlays on the specified destination. The overlays can be enumerated in front-to-back or back-to-front order.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

dwFlags

DDENUMOVERLAYZ_BACKTOFRONT

Enumerates overlays back to front.

DDENUMOVERLAYZ_FRONTTOBACK

Enumerates overlays front to back.

lpContext

Address of the user-defined context that will be passed to the callback function for each overlay surface.

lpfnCallback

Address of the **fnCallback** function that will be called for each surface being overlaid on this surface.

IDirectDrawSurface::Flip

```
HRESULT Flip(
    LPDIRECTDRAWSURFACE lpDDSurfaceTargetOverride,
    DWORD dwFlags);
```

Makes the surface memory associated with the DDSCAPS_BACKBUFFER surface become associated with the front buffer surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOTFLIPPABLE**
DDERR_NOFLIPHW	**DDERR_SURFACEBUSY**
DDERR_SURFACELOST	**DDERR_WASSTILLDRAWING**
DDERR_UNSUPPORTED	

lpDDSurfaceTargetOverride

Address of the DirectDrawSurface structure that will be flipped to. The default for this parameter is NULL, in which case **IDirectDrawSurface::Flip** cycles through the buffers in the order they are attached to each other. This parameter is only used as an override.

dwFlags

DDFLIP_WAIT

Typically, if the flip cannot be set up because the state of the display hardware is not appropriate, the error DDERR_WASSTILLDRAWING will return immediately and no flip will occur. Setting this flag causes **IDirectDrawSurface::Flip** to continue trying to flip if it receives the DDERR_WASSTILLDRAWING error from the HAL. **IDirectDrawSurface::Flip** will not return until the flipping operation has been successfully set up, or if another error, such as DDERR_SURFACEBUSY, is returned.

This method can only be called by a surface that has the DDSCAPS_FLIP and DDSCAPS_FRONTBUFFER values set. The display memory previously associated with the front buffer is associated with the back buffer. If there is more than one back buffer, a ring is formed and the surface memory buffers cycle one step through it every time **IDirectDrawSurface::Flip** is invoked.

The *lpDDSurfaceTargetOverride* parameter is used in rare cases when the back buffer is not the buffer that should become the front buffer. Typically this parameter is NULL.

The **IDirectDrawSurface::Flip** method will always be synchronized with the vertical blank.

See also **IDirectDrawSurface::GetFlipStatus**

IDirectDrawSurface::GetAttachedSurface

```
HRESULT GetAttachedSurface(LPDDSCAPS lpDDSCaps,
    LPLPDIRECTDRAWSURFACE FAR * lplpDDAttachedSurface);
```

Obtains the attached surface that has the specified capabilities.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOTFOUND	**DDERR_SURFACELOST**

lpDDSCaps
Address of a **DDSCAPS** structure that contains the hardware capabilities of the surface.

lplpDDAttachedSurface
Address of a pointer to a DirectDrawSurface that will be attached to the current DirectDrawSurface specified by *lpDDSurface* and has capabilities that match those specified by the *lpDDSCaps* parameter.

Attachments are used to connect multiple DirectDrawSurface objects into complex structures, like the ones needed to support 3D page flipping with z-buffers. This method will fail if more than one surface is attached that matches the capabilities requested. In this case, the application must use the

IDirectDrawSurface::EnumAttachedSurfaces method to obtain the non-unique attached surfaces.

IDirectDrawSurface::GetBltStatus

```
HRESULT GetBltStatus(DWORD dwFlags);
```

Obtains the blitter status. This method returns DD_OK if a blitter is present, DDERR_WASSTILLDRAWING if the blitter is busy, or DDERR_NOBLTHW if there is no blitter.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOBLTHW	**DDERR_SURFACEBUSY**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**
DDERR_WASSTILLDRAWING	

dwFlags

DDGBS_CANBLT

Inquires whether a blit involving this surface can occur immediately. Returns DD_OK if the blit can be completed.

DDGBS_ISBLTDONE

Inquires whether the blit is done. Returns DD_OK if the last blit on this surface has completed.

IDirectDrawSurface::GetCaps

```
HRESULT GetCaps(LPDDSCAPS lpDDSCaps);
```

Returns the capabilities of the surface. These capabilities are not necessarily related to the capabilities of the display device.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lpDDCaps

Address of a **DDCAPS** structure that will be filled in with the hardware capabilities of the surface.

IDirectDrawSurface::GetClipper

```
HRESULT GetClipper(
    LPDIRECTDRAWCLIPPER FAR * lplpDDClipper);
```

Returns the DirectDrawClipper associated with this surface. An error returns if no DirectDrawClipper is associated.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOCLIPPERATTACHED	

lplpDDClipper
Address of a pointer to the DirectDrawClipper associated with the surface.

See also **IDirectDrawSurface::SetClipper**

IDirectDrawSurface::GetColorKey

```
HRESULT GetColorKey(DWORD dwFlags,
    LPDDCOLORKEY lpDDColorKey);
```

Returns the color key value for the DirectDrawSurface object.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOCOLORKEYHW	**DDERR_NOCOLORKEY**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**

dwFlags
Determines which color key is requested.

DDCKEY_DESTBLT
Set if the structure specifies a color key or color space to be used as a destination color key for blit operations.

DDCKEY_DESTOVERLAY
Set if the structure specifies a color key or color space to be used as a destination color key for overlay operations.

DDCKEY_SRCBLT
Set if the structure specifies a color key or color space to be used as a source color key for blit operations.

DDCKEY_SRCOVERLAY
Set if the structure specifies a color key or color space to be used as a source color key for overlay operations.

lpDDColorKey
Address of the **DDCOLORKEY** structure that will be filled in with the current values for the specified color key of the DirectDrawSurface object.

See also **IDirectDrawSurface::SetColorKey**

IDirectDrawSurface::GetDC

```
HRESULT GetDC(HDC FAR * lphDC);
```

Creates a GDI-compatible hDC for the surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_DCALREADYCREATED	**DDERR_GENERIC**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDSURFACETYPE	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	**DDERR_WASSTILLDRAWING**

lphDC
Address of the returned hDC.

This method uses an internal version of the **IDirectDrawSurface::Lock** method to lock the surface, and the surface will remain locked until the **IDirectDrawSurface::ReleaseDC** method is called. For more information, see the description of the **IDirectDrawSurface::Lock** method.

See also **IDirectDrawSurface::ReleaseDC**, **IDirectDrawSurface::Lock**

IDirectDrawSurface2::GetDDInterface

```
HRESULT GetDDInterface(LPVOID FAR *lplpDD);
```

Returns an interface to the DirectDraw object that was used to create the surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lplpDD
Address of a pointer that will be filled with a valid DirectDraw pointer if the call succeeds.

To ensure COM compliance, this method is not part of the IDirectDrawSurface interface, but of the IDirectDrawSurface2 interface. To use this method, you must first query for the IDirectDrawSurface2 interface. For more information, see **IDirectDrawSurface2 Interface**.

IDirectDrawSurface::GetFlipStatus

```
HRESULT GetFlipStatus(DWORD dwFlags);
```

Indicates whether the surface has finished its flipping process. If the surface has not finished its flipping process, it returns DDERR_WASSTILLDRAWING.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDSURFACETYPE	**DDERR_SURFACEBUSY**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**
DDERR_WASSTILLDRAWING	

dwFlags

DDGFS_CANFLIP

Inquires whether this surface be flipped immediately. Returns DD_OK if the flip can be completed.

DDGFS_ISFLIPDONE

Inquires whether the flip done. Returns DD_OK if the last flip on this surface has completed.

See also **IDirectDrawSurface::Flip**

IDirectDrawSurface::GetOverlayPosition

```
HRESULT GetOverlayPosition(LPLONG lplX, LPLONG lplY);
```

Given a visible, active overlay surface (DDSCAPS_OVERLAY flag set), this method returns the display coordinates of the surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_INVALIDPOSITION**
DDERR_NOOVERLAYDEST	**DDERR_NOTAOVERLAYSURFACE**
DDERR_OVERLAYNOTVISIBLE	**DDERR_SURFACELOST**

lplX

Address of the x-display coordinate.

lplY

Address of the y-display coordinate.

See also **IDirectDrawSurface::SetOverlayPosition**, **IDirectDrawSurface::UpdateOverlay**

IDirectDrawSurface::GetPalette

```
HRESULT GetPalette(
    LPDIRECTDRAWPALETTE FAR * lplpDDPalette);
```

Returns the DirectDrawPalette structure associated with this surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOEXCLUSIVEMODE**
DDERR_NOPALETTEATTACHED	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

lplpDDPalette
Address of a pointer to a DirectDrawPalette structure. This pointer will be filled in with the address of the DirectDrawPalette structure associated with this surface. This will be set to NULL if no DirectDrawPalette is associated with this surface.

If no palette has been explicitly associated with this surface, it returns NULL for the associated palette. However, if this is the primary surface or a back buffer to the primary surface, it returns a pointer to the system palette if the primary surface is in 8-bpp mode.

See also **IDirectDrawSurface::SetPalette**

IDirectDrawSurface::GetPixelFormat

```
HRESULT GetPixelFormat(LPDDPIXELFORMAT lpDDPixelFormat);
```

Returns the color and pixel format of the surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDSURFACETYPE	

lpDDPixelFormat
Address of the **DDPIXELFORMAT** structure that will be filled in with a detailed description of the current pixel and color space format of the surface.

IDirectDrawSurface::GetSurfaceDesc

```
HRESULT GetSurfaceDesc(LPDDSURFACEDESC lpDDSurfaceDesc);
```

Returns a **DDSURFACEDESC** structure that describes the surface in its current condition.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure to be filled in with the current description of this surface.

See also **DDSURFACEDESC**

IDirectDrawSurface::Initialize

```
HRESULT Initialize(LPDIRECTDRAW lpDD,
    LPDDSURFACEDESC lpDDSurfaceDesc);
```

Initializes a DirectDrawSurface.

- Returns **DDERR_ALREADYINITIALIZED**.

lpDD
Address of the DirectDraw structure that represents the DirectDraw object.

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure to be filled in with the relevant details about the surface.

This method is provided for compliance with the Component Object Model (COM) protocol. Since the DirectDrawSurface object is initialized when it is created, calling this method will always result in the DDERR_ALREADYINITIALIZED return value.

See also **IDirectDrawSurface::AddRef**, **IDirectDrawSurface::QueryInterface**, **IDirectDrawSurface::Release**

IDirectDrawSurface::IsLost

```
HRESULT IsLost();
```

Determines if the surface memory associated with a DirectDrawSurface has been freed. If the memory has not been freed, this method will return DD_OK.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_SURFACELOST	

This method can be used to reallocate surface memory. When a DirectDrawSurface object loses its surface memory, most methods will return DDERR_SURFACELOST and perform no other function.

Surfaces can lose their memory when the mode of the display card is changed, or because an application received exclusive access to the display card and freed all of the surface memory currently allocated on the display card.

See also **IDirectDrawSurface::Restore**

IDirectDrawSurface::Lock

```
HRESULT Lock(LPRECT lpDestRect,
    LPDDSURFACEDESC lpDDSurfaceDesc,
    DWORD dwFlags, HANDLE hEvent);
```

Obtains a pointer to the surface memory.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_OUTOFMEMORY	**DDERR_SURFACEBUSY**
DDERR_SURFACELOST	**DDERR_WASSTILLDRAWING**

lpDestRect
Address of a RECT structure that identifies the region of surface that is being locked.

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure to be filled with the relevant details about the surface.

dwFlags

DDLOCK_EVENT

Triggers the event when **IDirectDrawSurface::Lock** can return the surface memory pointer requested. Set if an event handle is being passed to **IDirectDrawSurface::Lock**. If multiple locks of this type are placed on a surface, events will be triggered in FIFO order.

DDLOCK_READONLY

Indicates that the surface being locked will only be read from.

DDLOCK_SURFACEMEMORYPTR

Indicates that a valid memory pointer to the top of the specified rectangle should be returned. If no rectangle is specified, a pointer to the top of the surface is returned. This is the default.

DDLOCK_WAIT

Typically, if a lock cannot be obtained because a blit is in progress, a DDERR_WASSTILLDRAWING error will be returned immediately. If this flag is set, however, **IDirectDrawSurface::Lock** will retry until a lock is obtained or another error, such as DDERR_SURFACEBUSY, occurs.

DDLOCK_WRITEONLY

Indicates that the surface being locked will only be written to.

hEvent
Handle to a system event that is triggered when the surface is ready to be locked.

Once the pointer is obtained, the surface memory can be accessed by your application until the corresponding the **IDirectDrawSurface::Unlock** method is called. Once this occurs, the pointer to the surface memory is no longer valid.

It is illegal to blit from a region of a surface that is locked. If a blit is attempted on a locked surface, the blit will return either a DDERR_SURFACEBUSY or DDERR_LOCKEDSURFACES error value.

Typically, **IDirectDrawSurface::Lock** will return immediately with an error when a lock cannot be obtained because a blit is in progress. The DDLOCK_WAIT flag can be set to continue trying to obtain a lock.

To prevent VRAM from being lost during access to a surface, DirectDraw holds the Win16 lock between **IDirectDrawSurface::Lock** and **IDirectDrawSurface::Unlock** operations. The Win16 lock is the critical section that serializes access to GDI and USER. Although this technique allows direct access to display memory and prevents other applications from changing the mode during this access, it will stop Windows from running, so **IDirectDrawSurface::Lock**/**IDirectDrawSurface::Unlock** and **IDirectDrawSurface::GetDC**/**IDirectDrawSurface::ReleaseDC** periods should be kept short. Unfortunately, because Windows is stopped, GUI debuggers cannot be used in between **IDirectDrawSurface::Lock**/**IDirectDrawSurface::Unlock** or **IDirectDrawSurface::GetDC**/**IDirectDrawSurface::ReleaseDC** operations.

See also **IDirectDrawSurface::Unlock**, **IDirectDrawSurface::GetDC**, **IDirectDrawSurface::ReleaseDC**

IDirectDrawSurface2::PageLock

```
HRESULT PageLock(DWORD dwFlags);
```

Prevents a system memory surface from being paged out while a blit using DMA transfers to or from system memory is in progress. A lock count is maintained for each surface and is incremented each time **IDirectDrawSurface2::PageLock** is called for that surface. The count is decremented when **IDirectDrawSurface2::PageUnlock** is called. When the count reaches 0, the memory is unlocked and can then be paged by the operating system.

Not e The performance of the operating system could be negatively affected if too much memory is locked.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_CANTPAGELOCK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**

dwFlags
This parameter is not used at this time and must be set to 0.

This method only works on system memory surfaces; it will not page lock a display memory surface or an emulated primary surface. If this method is called on a display memory surface, it will do nothing except return DD_OK.

To ensure COM compliance, this method is not a part of the IDirectDrawSurface interface, but belongs to the IDirectDrawSurface2 interface. To use this method, you must first query for the IDirectDrawSurface2 interface. For more information, see **IDirectDrawSurface2 Interface**.

See also **IDirectDrawSurface2::PageUnlock**

IDirectDrawSurface2::PageUnlock

```
HRESULT PageUnlock(DWORD dwFlags);
```

Unlocks a system memory surface, allowing it to be paged out. A lock count is maintained for each surface and is incremented each time **IDirectDrawSurface2::PageLock** is called for that surface. The count is decremented when **IDirectDrawSurface2::PageUnlock** is called. When the count reaches 0, the memory is unlocked and can then be paged by the operating system.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_CANTPAGEUNLOCK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOTPAGELOCKED**
DDERR_SURFACELOST	

dwFlags
This parameter is not used at this time and must be set to 0.

This method only works on system memory surfaces; it will not page unlock a display memory surface or an emulated primary surface. If this method is called on a display memory surface, it will do nothing except return DD_OK.

To ensure COM compliance, this method is not a part of the IDirectDrawSurface interface, but belongs to the IDirectDrawSurface2 interface. To use this method, you must first query for the IDirectDrawSurface2 interface. For more information, see **IDirectDrawSurface2 Interface**.

See also **IDirectDrawSurface2::PageLock**

IDirectDrawSurface::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID FAR * ppvObj);
```

Determines if the DirectDrawSurface object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_OUTOFMEMORY	

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that will be filled with the interface pointer if the query is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirectDrawSurface::QueryInterface** method allows DirectDrawSurface objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

See also **IDirectDrawSurface::AddRef**, **IDirectDrawSurface::Initialize**, **IDirectDrawSurface::Release**

IDirectDrawSurface::Release

```
ULONG Release();
```

Decreases the reference count of the DirectDrawSurface object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

The DirectDrawSurface object deallocates itself when its reference count reaches 0. Use the **IDirectDrawSurface::AddRef** method to increase the reference count of the object by 1.

See also **IDirectDrawSurface::AddRef**, **IDirectDrawSurface::Initialize**, **IDirectDrawSurface::QueryInterface**

IDirectDrawSurface::ReleaseDC

```
HRESULT ReleaseDC(HDC hDC);
```

Releases the hDC previously obtained with the **IDirectDrawSurface::GetDC** method.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

hDC
The hDC previously obtained by **IDirectDrawSurface::GetDC**.

This method also unlocks the surface previously locked when the **IDirectDrawSurface::GetDC** method was called.

See also **IDirectDrawSurface::GetDC**

IDirectDrawSurface::Restore

```
HRESULT Restore();
```

Restores a surface that has been lost. This occurs when the surface memory associated with the DirectDrawSurface object has been freed.

- Returns DD_OK is successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INCOMPATIBLEPRIMARY**
DDERR_IMPLICITLYCREATED	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOEXCLUSIVEMODE**
DDERR_OUTOFMEMORY	**DDERR_UNSUPPORTED**
DDERR_WRONGMODE	

Surfaces can be lost because the display card's mode was changed or because an application received exclusive access to the display card and freed all of the surface memory currently allocated on the card. When a DirectDrawSurface object loses its surface memory, many methods will return DDERR_SURFACELOST and perform no other function. The **IDirectDrawSurface::Restore** method will reallocate surface memory and reattach it to the DirectDrawSurface object.

A single call to this method will restore a DirectDrawSurface's associated implicit surfaces (back buffers, and so on). An attempt to restore an implicitly created surface will result in an error. **IDirectDrawSurface::Restore** will not work across explicit attachments created using the **IDirectDrawSurface::AddAttachedSurface** method — each of these surfaces must be restored individually.

See also **IDirectDrawSurface::IsLost**, **IDirectDrawSurface::AddAttachedSurface**

IDirectDrawSurface::SetClipper

```
HRESULT SetClipper(LPDIRECTDRAWCLIPPER lpDDClipper);
```

Attaches a DirectDrawClipper object to a DirectDrawSurface object.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDSURFACETYPE	**DDERR_NOCLIPPERATTACHED**

lpDDClipper
Address of the DirectDrawClipper structure representing the DirectDrawClipper that will be attached to the DirectDrawSurface. If NULL, the current clipper will be detached.

This method is primarily used by surfaces that are being overlaid on or blitted to the primary surface. However, it can be used on any surface. Once a DirectDrawClipper has been attached and a clip list is associated with it, the clipper object will be used for the **IDirectDrawSurface::Blt**, **IDirectDrawSurface::BltBatch**, and **IDirectDrawSurface::UpdateOverlay** operations involving the parent DirectDrawSurface. This method can also detach a DirectDrawSurface's current clipper.

See also **IDirectDrawSurface::GetClipper**

IDirectDrawSurface::SetColorKey

```
HRESULT SetColorKey(DWORD dwFlags,
    LPDDCOLORKEY lpDDColorKey);
```

Sets the color key value for the DirectDrawSurface object if the hardware supports color keys on a per surface basis.

- Returns DD_OK is successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_INVALIDSURFACETYPE**
DDERR_NOOVERLAYHW	**DDERR_NOTAOVERLAYSURFACE**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**
DDERR_WASSTILLDRAWING	

dwFlags
Determines which color key is requested.

DDCKEY_COLORSPACE

Set if the structure contains a color space. Not set if the structure contains a single color key.

DDCKEY_DESTBLT

Set if the structure specifies a color key or color space to be used as a destination color key for blit operations.

DDCKEY_DESTOVERLAY

Set if the structure specifies a color key or color space to be used as a destination color key for overlay operations.

DDCKEY_SRCBLT

Set if the structure specifies a color key or color space to be used as a source color key for blit operations.

DDCKEY_SRCOVERLAY

Set if the structure specifies a color key or color space to be used as a source color key for overlay operations.

lpDDColorKey

Address of the **DDCOLORKEY** structure that has the new color key values for the DirectDrawSurface object.

See also **IDirectDrawSurface::GetColorKey**

IDirectDrawSurface::SetOverlayPosition

```
HRESULT SetOverlayPosition(LONG lX, LONG lY);
```

Changes the display coordinates of an overlay surface.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**
DDERR_UNSUPPORTED	

lX

New x-display coordinate.

lY

New y-display coordinate.

See also **IDirectDrawSurface::GetOverlayPosition**, **IDirectDrawSurface::UpdateOverlay**

IDirectDrawSurface::SetPalette

```
HRESULT SetPalette(LPDIRECTDRAWPALETTE lpDDPalette);
```

Attaches the specified DirectDrawPalette to a surface. The surface will use this palette for all subsequent operations. The palette change takes place immediately, without regard to refresh timing.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_INVALIDSURFACETYPE**
DDERR_NOEXCLUSIVEMODE	**DDERR_NOPALETTEATTACHED**
DDERR_NOPALETTEHW	**DDERR_NOT8BITCOLOR**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**

lpDDPalette
Address of the DirectDrawPalette structure that this surface should use for future operations.

See also **IDirectDrawSurface::GetPalette**, **IDirectDraw::CreatePalette**

IDirectDrawSurface::Unlock

```
HRESULT Unlock(LPVOID lpSurfaceData);
```

Notifies DirectDraw that the direct surface manipulations are complete.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_INVALIDRECT**
DDERR_NOTLOCKED	**DDERR_SURFACELOST**

lpSurfaceData
Address of a pointer returned by the **IDirectDrawSurface::Lock** method. Since it is possible to call **IDirectDrawSurface::Lock** multiple times for the same surface with different destination rectangles, this pointer ties the calls to the **IDirectDrawSurface::Lock** and **IDirectDrawSurface::Unlock** methods together.

See also **IDirectDrawSurface::Lock**

IDirectDrawSurface::UpdateOverlay

```
HRESULT UpdateOverlay(LPRECT lpSrcRect,
    LPDIRECTDRAWSURFACE lpDDDestSurface,
    LPRECT lpDestRect, DWORD dwFlags,
    LPDDOVERLAYFX lpDDOverlayFx);
```

Repositions or modifies the visual attributes of an overlay surface. These surfaces must have the DDSCAPS_OVERLAY value set.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_HEIGHTALIGN**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDRECT	**DDERR_INVALIDSURFACETYPE**
DDERR_NOSTRETCHHW	**DDERR_NOTAOVERLAYSURFACE**
DDERR_SURFACELOST	**DDERR_UNSUPPORTED**
DDERR_XALIGN	

lpSrcRect
Address of a RECT structure that defines the x, y, width, and height of the region on the source surface being used as the overlay.

lpDDDestSurface
Address of the DirectDrawSurface structure that represents the DirectDrawSurface. This is the surface that is being overlaid.

lpDestRect
Address of a RECT structure that defines the x, y, width, and height of the region on the destination surface that the overlay should be moved to.

dwFlags

DDOVER_ADDDIRTYRECT
Adds a dirty rectangle to an emulated overlaid surface.

DDOVER_ALPHADEST
Uses the alpha information in pixel format or the alpha channel surface attached to the destination surface as the alpha channel for the destination overlay.

DDOVER_ALPHADESTCONSTOVERRIDE
Uses the **dwAlphaDestConst** member in the **DDOVERLAYFX** structure as the destination alpha channel for this overlay.

DDOVER_ALPHADESTNEG
The NEG suffix indicates that the destination surface becomes more transparent as the alpha value increases. (0 is opaque).

DDOVER_ALPHADESTSURFACEOVERRIDE
Uses the **lpDDSAlphaDest** member in the **DDOVERLAYFX** structure as the alpha channel destination for this overlay.

DDOVER_ALPHAEDGEBLEND
Uses the **dwAlphaEdgeBlend** member in the **DDOVERLAYFX** structure as the alpha channel for the edges of the image that border the color key colors.

DDOVER_ALPHASRC

Uses the alpha information in pixel format or the alpha channel surface attached to the source surface as the source alpha channel for this overlay.

DDOVER_ALPHASRCCONSTOVERRIDE

Uses the **dwAlphaSrcConst** member in the **DDOVERLAYFX** structure as the source alpha channel for this overlay.

DDOVER_ALPHASRCNEG

The NEG suffix indicates that the source surface becomes more transparent as the alpha value increases.

DDOVER_ALPHASRCSURFACEOVERRIDE

Uses the **lpDDSAlphaSrc** member in the **DDOVERLAYFX** structure as the alpha channel source for this overlay.

DDOVER_DDFX

Uses the overlay FX flags to define special overlay FX.

DDOVER_HIDE

Turns this overlay off.

DDOVER_KEYDEST

Uses the color key associated with the destination surface.

DDOVER_KEYDESTOVERRIDE

Uses the **dckDestColorkey** member in the **DDOVERLAYFX** structure as the color key for the destination surface.

DDOVER_KEYSRC

Uses the color key associated with the source surface.

DDOVER_KEYSRCOVERRIDE

Uses the **dckSrcColorkey** member in the **DDOVERLAYFX** structure as the color key for the source surface.

DDOVER_SHOW

Turns this overlay on.

DDOVER_ZORDER

Uses the **dwZOrderFlags** member in the **DDOVERLAYFX** structure as the z-order for the display of this overlay. The **lpDDSRelative** member will be used if the **dwZOrderFlags** member is set to either DDOVERZ_INSERTINBACKOF or DDOVERZ_INSERTINFRONTOF.

lpDDOverlayFx

See the **DDOVERLAYFX** structure.

IDirectDrawSurface::UpdateOverlayDisplay

```
HRESULT UpdateOverlayDisplay(DWORD dwFlags);
```

Repaints the rectangles in the dirty rectangle list of all active overlays. This clears the dirty rectangle list. This method is for software emulation only—it does nothing if the hardware supports overlays.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_UNSUPPORTED	**DDERR_INVALIDSURFACETYPE**

dwFlags
Specifies the type of update to perform.

DDOVER_REFRESHDIRTYRECTS
Updates the overlay display using the list of dirty rectangles previously constructed for this destination. This clears the dirty rectangle list.

DDOVER_REFRESHALL
Ignores the dirty rectangle list and updates the overlay display completely. This clears the dirty rectangle list.

See also **IDirectDrawSurface::AddOverlayDirtyRect**

IDirectDrawSurface::UpdateOverlayZOrder

```
HRESULT UpdateOverlayZOrder(DWORD dwFlags,
    LPDIRECTDRAWSURFACE lpDDSReference);
```

Sets the z-order of an overlay. The z-order determines which overlay should be occluded when multiple overlays are displayed simultaneously. Overlay positions are all relative to other overlays — there is no true z-value for them.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOTAOVERLAYSURFACE	

dwFlags

DDOVERZ_INSERTINBACKOF
Inserts this overlay in the overlay chain behind the reference overlay.

DDOVERZ_INSERTINFRONTOF
Inserts this overlay in the overlay chain in front of the reference overlay.

DDOVERZ_MOVEBACKWARD
Moves this overlay one position backward in the overlay chain.

DDOVERZ_MOVEFORWARD
Moves this overlay one position forward in the overlay chain.

DDOVERZ_SENDTOBACK

Moves this overlay to the back of the overlay chain.

DDOVERZ_SENDTOFRONT

Moves this overlay to the front of the overlay chain.

lpDDSReference
Address of the DirectDrawSurface structure that represents the DirectDrawSurface to be used as a relative position in the overlay chain. This parameter is needed only for DDOVERZ_INSERTINBACKOF and DDOVERZ_INSERTINFRONTOF.

See also **IDirectDrawSurface::EnumOverlayZOrders**

IDirectDrawPalette Interface

The DirectDrawPalette object is provided to enable direct manipulation of 16- and 256-color palettes. DirectDrawPalette reserves entries 0 through 255 for 256-color palettes; however, it reserves no entries for 16-color palettes. It allows direct manipulation of the palette table as a table. This table can contain 16- or 24-bit RGB entries representing the colors associated with each of the indexes. For 16-color palettes, the table can also contain indexes to another 256-color palette.

Entries in these tables can be retrieved with the **IDirectDrawPalette::GetEntries** method and changed with the **IDirectDrawPalette::SetEntries** method. The **IDirectDrawPalette::SetEntries** method has a *dwFlags* parameter that specifies when the changes to the palette should take effect.

DirectDrawPalette objects are usually attached to DirectDrawSurface objects.

Two approaches can be used to provide straightforward palette animation using DirectDrawPalette objects. The first approach involves changing the palette entries that correspond to the colors that need to be animated. This can be done with a single call to the **IDirectDrawPalette::SetEntries** method. The second approach requires two DirectDrawPalette objects. The animation is performed by attaching first one object then the other to the DirectDrawSurface. This can be done using the **IDirectDrawSurface::SetPalette** method.

IDirectDrawPalette Interface Method Groups

Applications use the methods of the IDirectDrawPalette interface to create DirectDrawPalette objects and work with system-level variables. The methods can be organized into the following groups:

Allocating memory	**Initialize**
IUnknown	**AddRef**
	QueryInterface

	Release
Palette capabilities	**GetCaps**
Palette entries	**GetEntries** **SetEntries**

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the DirectDrawPalette object without affecting the functionality of the original interface.

IDirectDrawPalette::AddRef

```
ULONG AddRef();
```

Increases the reference count of the DirectDrawPalette object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

When the DirectDraw object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirectDrawPalette::Release** method to decrease the reference count of the object by 1.

See also **IDirectDrawPalette::Initialize**, **IDirectDrawPalette::QueryInterface**, **IDirectDrawPalette::Release**

IDirectDrawPalette::GetCaps

```
HRESULT GetCaps(LPDWORD lpdwCaps);
```

Returns the capabilities of this palette object.

- Returns DD_OK if successful, or one of the following error values otherwise:

  DDERR_INVALIDOBJECT **DDERR_INVALIDPARAMS**

lpdwCaps
  Flags from the **dwPalCaps** member of the **DDCAPS** structure that define palette capabilities.

  DDPCAPS_4BIT

  DDPCAPS_8BIT

DDPCAPS_8BITENTRIES

DDPCAPS_ALLOW256

DDPCAPS_PRIMARYSURFACE

DDPCAPS_PRIMARYSURFACELEFT

DDPCAPS_VSYNC

IDirectDrawPalette::GetEntries

```
HRESULT GetEntries(DWORD dwFlags, DWORD dwBase,
    DWORD dwNumEntries, LPPALETTEENTRY lpEntries );
```

Queries palette values from a DirectDrawPalette.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOTPALETTIZED	

dwFlags
This parameter is not used at this time and must be set to 0.

dwBase
Indicates the start of the entries that should be retrieved sequentially.

dwNumEntries
Indicates the number of palette entries *lpEntries* has room for. The colors of each palette entry will be returned in sequence, from *dwStartingEntry* through *dwCount* -1.

lpEntries
Address of the palette entries. The palette entries are one byte each if the DDPCAPS_8BITENTRIES flag is set, and four bytes otherwise. Each field is a color description.

See also **IDirectDrawPalette::SetEntries**

IDirectDrawPalette::Initialize

```
HRESULT Initialize(LPDIRECTDRAW lpDD, DWORD dwFlags,
    LPPALETTEENTRY lpDDColorTable);
```

Initializes the DirectDrawPalette object.

- Returns **DDERR_ALREADYINITIALIZED**.

lpDD
Address of the DirectDraw structure that represents the DirectDraw object.

dwFlags
This parameter is not used at this time and must be set to 0.

lpDDColorTable
This parameter is not used at this time and must be set to 0.

This method is provided for compliance with the Component Object Model (COM) protocol. Since the DirectDrawPalette object is initialized when it is created, calling this method will always result in the DDERR_ALREADYINITIALIZED return value.

See also **IDirectDrawPalette::AddRef**, **IDirectDrawPalette::QueryInterface**, **IDirectDrawPalette::Initialize**

IDirectDrawPalette::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID FAR* ppvObj);
```

Determines if the DirectDrawPalette object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that receives the interface pointer if the request is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirectDrawPalette::QueryInterface** method allows DirectDrawPalette objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

See also **IDirectDrawPalette::AddRef**, **IDirectDrawPalette::Initialize**, **IDirectDrawPalette::Release**

IDirectDrawPalette::Release

```
ULONG Release();
```

Decreases the reference count of the DirectDrawPalette object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

The DirectDrawPalette object deallocates itself when its reference count reaches 0. Use the **IDirectDrawPalette::AddRef** method to increase the reference count of the object by 1.

See also **IDirectDrawPalette::AddRef**, **IDirectDrawPalette::Initialize**, **IDirectDrawPalette::QueryInterface**

IDirectDrawPalette::SetEntries

```
HRESULT SetEntries(DWORD dwFlags,
    DWORD dwStartingEntry, DWORD dwCount,
    LPPALETTEENTRY lpEntries);
```

Changes entries in a DirectDrawPalette immediately. The palette must be attached to a surface using the **IDirectDrawSurface::SetPalette** method before **IDirectDrawPalette::SetEntries** can be used.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOPALETTEATTACHED	**DDERR_NOTPALETTIZED**
DDERR_UNSUPPORTED	

dwFlags
This parameter is not used at this time and must be set to 0.

dwStartingEntry
Specifies the first entry to be set.

dwCount
Specifies the number of palette entries to be changed.

lpEntries
Address of the palette entries. The palette entries are one byte each if the DDPCAPS_8BITENTRIES flag is set and four bytes otherwise. Each field is a color description.

See also **IDirectDrawPalette::GetEntries**, **IDirectDrawSurface::SetPalette**

IDirectDrawClipper Interface

IDirectDrawClipper Interface Method Groups

Applications use the methods of the IDirectDraw interface to create DirectDraw objects and work with system-level variables. The methods can be organized into the following groups:

Allocating memory	**Initialize**

Clip lists	**IsClipListChanged**
	SetClipList
	SetHWnd
Handles	**GetClipList**
	GetHWnd
IUnknown	**AddRef**
	QueryInterface
	Release

All COM interfaces inherit the **IUnknown** interface methods, which are listed in the "IUnknown" group above. These three methods allow additional interfaces to be added to the DirectDrawClipper object without affecting the functionality of the original interface.

IDirectDrawClipper::AddRef

```
ULONG AddRef();
```

Increases the reference count of the DirectDrawClipper object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

When the DirectDraw object is created, its reference count is set to 1. Every time an application obtains an interface to the object or calls the **AddRef** method, the object's reference count is increased by 1. Use the **IDirectDrawClipper::Release** method to decrease the reference count of the object by 1.

See also **IDirectDraw::CreateClipper**, **IDirectDrawClipper::Initialize**, **IDirectDrawClipper::QueryInterface**, **IDirectDrawClipper::Release**

IDirectDrawClipper::GetClipList

```
HRESULT GetClipList(LPRECT lpRect,
    LPRGNDATA lpClipList, LPDWORD lpdwSize);
```

Returns a copy of the clip list associated with a DirectDrawClipper object. A subset of the clip list can be selected by passing a rectangle that clips the clip list.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_GENERIC	**DDERR_INVALIDCLIPLIST**
DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_NOCLIPLIST	**DDERR_REGIONTOOSMALL**

lpRect
Address of a rectangle that will be used to clip the clip list.

lpClipList
Address of an RGNDATA structure that will contain the resulting copy of the clip list.

lpdwSize
Set by **IDirectDrawClipper::GetClipList** to indicate the size of the resulting clip list.

See also **IDirectDrawClipper::SetClipList**

IDirectDrawClipper::GetHWnd

```
HRESULT GetHWnd(HWND FAR * lphWnd);
```

Returns the hWnd previously associated with this DirectDrawClipper object by the **IDirectDrawClipper::SetHWnd** method.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lphWnd
Address of the hWnd previously associated with this DirectDrawClipper by the **IDirectDrawClipper::SetHWnd** method.

See also **IDirectDrawClipper::SetHWnd**

IDirectDrawClipper::Initialize

```
HRESULT Initialize(LPDIRECTDRAW lpDD, DWORD dwFlags);
```

Initializes a DirectDrawClipper.

- Returns **DDERR_ALREADYINITIALIZED**.

lpDD
Address of the DirectDraw structure that represents the DirectDraw object.

dwFlags
This parameter is not used at this time and must be set to 0.

This method is provided for compliance with the Component Object Model (COM) protocol. Since the DirectDrawClipper object is initialized when it is created, calling this method will always result in the DDERR_ALREADYINITIALIZED return value.

See also **IDirectDrawClipper::AddRef**, **IDirectDraw::CreateClipper**, **IDirectDrawClipper::QueryInterface**, **IDirectDrawClipper::Release**

IDirectDrawClipper::IsClipListChanged

```
HRESULT IsClipListChanged(BOOL FAR * lpbChanged);
```

Monitors the status of the clip list if an hWnd is associated with a DirectDrawClipper.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

lpbChanged
Address of a Boolean value set to TRUE if the clip list has changed.

IDirectDrawClipper::QueryInterface

```
HRESULT QueryInterface(REFIID riid, LPVOID FAR * ppvObj);
```

Determines if the DirectDrawClipper object supports a particular COM interface. If it does, the system increases the reference count for the object, and the application can begin using that interface immediately. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**

riid
Reference identifier of the interface being requested.

ppvObj
Address of a pointer that receives the interface pointer if the request is successful.

If the application does not need to use the interface retrieved by a call to this method, it must call the **Release** method for that interface to free it. The **IDirectDrawClipper::QueryInterface** method allows DirectDrawClipper objects to be extended by Microsoft and third parties without interfering with each other's existing or future functionality.

See also **IDirectDrawClipper::AddRef**, **IDirectDrawClipper::Release**, **IDirectDrawClipper::Initialize**

IDirectDrawClipper::Release

```
ULONG Release();
```

Decreases the reference count of the DirectDrawClipper object by 1. This method is part of the **IUnknown** interface inherited by DirectDraw.

- Returns the new reference count of the object.

The DirectDrawClipper object deallocates itself when its reference count reaches 0. Use the **IDirectDrawClipper::AddRef** method to increase the reference count of the object by 1.

See also **IDirectDrawClipper::AddRef**, **IDirectDrawClipper::Initialize**, **IDirectDrawClipper::QueryInterface**

IDirectDrawClipper::SetClipList

```
HRESULT SetClipList(LPRGNDATA lpClipList, DWORD dwFlags);
```

Sets or deletes the clip list used by the **IDirectDrawSurface::Blt**, **IDirectDrawSurface::BltBatch**, and **IDirectDrawSurface::UpdateOverlay** methods on surfaces to which the parent DirectDrawClipper is attached.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_CLIPPERISUSINGHWND**
DDERR_INVALIDPARAMS	**DDERR_INVALIDCLIPLIST**
DDERR_OUTOFMEMORY	

lpClipList
Either an address to a valid RGNDATA structure or NULL. If there is an existing clip list associated with the DirectDrawClipper and this value is NULL, the clip list will be deleted.

dwFlags
This parameter is not used at this time and must be set to 0.

The clip list is a series of rectangles that describes the visible areas of the surface. The clip list cannot be set if an hWnd is already associated with the DirectDrawClipper object. Note that the **IDirectDrawSurface::BltFast** method cannot clip.

See also **IDirectDrawClipper::GetClipList**, **IDirectDrawSurface::Blt**, **IDirectDrawSurface::BltFast**, **IDirectDrawSurface::BltBatch**, **IDirectDrawSurface::UpdateOverlay**

IDirectDrawClipper::SetHWnd

```
HRESULT SetHWnd(DWORD dwFlags, HWND hWnd);
```

Sets the hWnd that will obtain the clipping information.

- Returns DD_OK if successful, or one of the following error values otherwise:

DDERR_INVALIDOBJECT	**DDERR_INVALIDPARAMS**
DDERR_INVALIDCLIPLIST	**DDERR_OUTOFMEMORY**

dwFlags
: This parameter is not used at this time and must be set to 0.

hWnd
: The hWnd that obtains the clipping information.

See also **IDirectDrawClipper::GetHWnd**

Structures

DDBLTBATCH

```
typedef struct _DDBLTBATCH{
    LPRECT              lprDest;
    LPDIRECTDRAWSURFACE lpDDSSrc;
    LPRECT              lprSrc;
    DWORD               dwFlags;
    LPDDBLTFX           lpDDBltFx;
} DDBLTBATCH,FAR *LPDDBLTBATCH;
```

Passes blit operations to the **IDirectDrawSurface::BltBatch** method.

lprDest
: Address of a RECT structure that defines the destination for the blit.

lpDDSSrc
: Address of a DirectDrawSurface that will be the source of the blit.

lprSrc
: Address of a RECT structure that defines the source rectangle of the blit.

dwFlags
: Specifies the optional control flags.

DDBLT_ALPHADEST
: Uses either the alpha information in the pixel format or the alpha channel surface attached to the destination surface as the alpha channel for this blit.

DDBLT_ALPHADESTCONSTOVERRIDE

Uses the **dwAlphaDestConst** member in the **DDBLTFX** structure as the alpha channel for the destination surface for this blit.

DDBLT_ALPHADESTNEG

The NEG suffix indicates that the destination surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHADESTSURFACEOVERRIDE

Uses the **lpDDSAlphaDest** member in the **DDBLTFX** structure as the alpha channel for the destination surface for this blit.

DDBLT_ALPHAEDGEBLEND

Uses the **dwAlphaEdgeBlend** member in the **DDBLTFX** structure as the alpha channel for the edges of the image that border the colors of the color key.

DDBLT_ALPHASRC

Uses either the alpha information in the pixel format or the alpha channel surface attached to the source surface as the alpha channel for this blit.

DDBLT_ALPHASRCCONSTOVERRIDE

Uses the **dwAlphaSrcConst** member in the **DDBLTFX** structure as the alpha channel for the source for this blit.

DDBLT_ALPHASRCNEG

The NEG suffix indicates that the source surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHASRCSURFACEOVERRIDE

Uses the **lpDDSAlphaSrc** member in the **DDBLTFX** structure as the alpha channel for the source for this blit.

DDBLT_ASYNC

Processes this blit asynchronously through the FIFO hardware in the order received. If there is no room in the FIFO hardware, the call fails.

DDBLT_COLORFILL

Uses the **dwFillColor** member in the **DDBLTFX** structure as the RGB color with which to fill the destination rectangle on the destination surface.

DDBLT_DDFX

Uses the **dwDDFX** member in the **DDBLTFX** structure to specify the effects to be used for this blit.

DDBLT_DDROPS

Uses the **dwDDROPS** member in the **DDBLTFX** structure to specify the raster operations that are not part of the Win32 API.

DDBLT_KEYDEST

Uses the color key associated with the destination surface.

DDBLT_KEYDESTOVERRIDE

Uses the **dckDestColorkey** member in the **DDBLTFX** structure as the color key for the destination surface.

DDBLT_KEYSRC

Uses the color key associated with the source surface.

DDBLT_KEYSRCOVERRIDE

Uses the **dckSrcColorkey** member in the **DDBLTFX** structure as the color key for the source surface.

DDBLT_ROP

Uses the **dwROP** member in the **DDBLTFX** structure for the raster operation for this blit. The raster operations are the same as those defined in the Win32 API.

DDBLT_ROTATIONANGLE

Uses the **dwRotationAngle** member in the **DDBLTFX** structure as the rotation angle (specified in 1/100th of a degree) for the surface.

DDBLT_ZBUFFER

A z-buffered blit using the z-buffers attached to the source and destination surfaces and the **dwZBufferOpCode** member in the **DDBLTFX** structure as the z-buffer opcode.

DDBLT_ZBUFFERDESTCONSTOVERRIDE

A z-buffered blit using the **dwZDestConst** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the destination.

DDBLT_ZBUFFERDESTOVERRIDE

A z-buffered blit using the **lpDDSZBufferDest** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the destination.

DDBLT_ZBUFFERSRCCONSTOVERRIDE

A z-buffered blit using the **dwZSrcConst** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the source.

DDBLT_ZBUFFERSRCOVERRIDE

A z-buffered blit using the **lpDDSZBufferSrc** and **dwZBufferOpCode** members in the **DDBLTFX** structure as the z-buffer and z-buffer opcode, respectively, for the source.

lpDDBltFx

Address of a **DDBLTFX** structure specifying additional blit effects.

DDBLTFX

```
typedef struct _DDBLTFX{
    DWORD  dwSize;
    DWORD  dwDDFX;
    DWORD  dwROP;
    DWORD  dwDDROP;
    DWORD  dwRotationAngle;
    DWORD  dwZBufferOpCode;
    DWORD  dwZBufferLow;
    DWORD  dwZBufferHigh;
    DWORD  dwZBufferBaseDest;
    DWORD  dwZDestConstBitDepth;
union
{
        DWORD                dwZDestConst;
        LPDIRECTDRAWSURFACE  lpDDSZBufferDest;
};
    DWORD   dwZSrcConstBitDepth;
union
{
        DWORD                dwZSrcConst;
        LPDIRECTDRAWSURFACE  lpDDSZBufferSrc;

};
    DWORD  dwAlphaEdgeBlendBitDepth;
    DWORD  dwAlphaEdgeBlend;
    DWORD  dwReserved;
    DWORD  dwAlphaDestConstBitDepth;
union
{
        DWORD                dwAlphaDestConst;
        LPDIRECTDRAWSURFACE  lpDDSAlphaDest;
};
    DWORD  dwAlphaSrcConstBitDepth;
union
{
        DWORD                dwAlphaSrcConst;
        LPDIRECTDRAWSURFACE  lpDDSAlphaSrc;
};
union
{
        DWORD                 dwFillColor;
        DWORD                 dwFillDepth;

        LPDIRECTDRAWSURFACE  lpDDSPattern;
};
DDCOLORKEY  ddckDestColorkey;
DDCOLORKEY  ddckSrcColorkey;
} DDBLTFX,FAR* LPDDBLTFX;
```

Passes raster operations, effects, and override information to the **IDirectDrawSurface::Blt** method. It is also part of the **DDBLTBATCH** structure used with the **IDirectDrawSurface::BltBatch** method.

dwSize
Size of the structure. This must be initialized before the structure is used.

dwDDFX
Specifies the type of FX operations.

DDBLTFX_ARITHSTRETCHY
Uses arithmetic stretching along the y-axis for this blit.

DDBLTFX_MIRRORLEFTRIGHT
Turns the surface on its y-axis. This blit mirrors the surface from left to right.

DDBLTFX_MIRRORUPDOWN
Turns the surface on its x-axis. This blit mirrors the surface from top to bottom.

DDBLTFX_NOTEARING
Schedules this blit to avoid tearing.

DDBLTFX_ROTATE180
Rotates the surface 180 degrees clockwise during this blit.

DDBLTFX_ROTATE270
Rotates the surface 270 degrees clockwise during this blit.

DDBLTFX_ROTATE90
Rotates the surface 90 degrees clockwise during this blit.

DDBLTFX_ZBUFFERBASEDEST
Adds the **dwZBufferBaseDest** member to each of the source z-values before comparing them with the destination z-values during this z-blit.

DDBLTFX_ZBUFFERRANGE
Uses the **dwZBufferLow** and **dwZBufferHigh** members as range values to specify limits to the bits copied from a source surface during this z-blit.

dwROP
Specifies the Win32 raster operations.

dwDDROP
Specifies the DirectDraw raster operations.

dwRotationAngle
Rotation angle for the blit.

dwZBufferOpCode
Z-buffer compares.

dwZBufferLow
Low limit of a z-buffer.

dwZBufferHigh
High limit of a z-buffer.

dwZBufferBaseDest
Destination base value of a z-buffer.

dwZDestConstBitDepth
Bit depth of the destination z-constant.

dwZDestConst
Constant used as the z-buffer destination.

lpDDSZBufferDest
Surface used as the z-buffer destination.

dwZSrcConstBitDepth
Bit depth of the source z-constant.

dwZSrcConst
Constant used as the z-buffer source.

lpDDSZBufferSrc
Surface used as the z-buffer source.

dwAlphaEdgeBlend
Alpha used for edge blending.

dwAlphaEdgeBlendBitDepth
Bit depth of the constant for an alpha edge blend.

dwReserved
Reserved for future use.

dwAlphaDestConstBitDepth
Bit depth of the destination alpha constant.

dwAlphaDestConst
Constant used as the alpha channel destination.

lpDDSAlphaDest
Surface used as the alpha channel destination.

dwAlphaSrcConstBitDepth
Bit depth of the source alpha constant.

dwAlphaSrcConst
Constant used as the alpha channel source.

lpDDSAlphaSrc
Surface used as the alpha channel source.

dwFillColor
Color used to fill a surface when DDBLT_COLORFILL is specified. This value can be either an RGB triple or a palette index, depending on the surface type.

dwFillDepth
Depth value for the z-buffer.

lpDDSPattern
Surface to use as a pattern. The pattern can be used in certain blit operations that combine a source and a destination.

ddckDestColorkey
Destination color key override.

ddckSrcColorkey
Source color key override.

DDCAPS

```
typedef struct _DDCAPS{
    DWORD    dwSize;
    DWORD    dwCaps;
    DWORD    dwCaps2;
    DWORD    dwCKeyCaps;
    DWORD    dwFXCaps;
    DWORD    dwFXAlphaCaps;
    DWORD    dwPalCaps;
    DWORD    dwSVCaps;
    DWORD    dwAlphaBltConstBitDepths;
    DWORD    dwAlphaBltPixelBitDepths;
    DWORD    dwAlphaBltSurfaceBitDepths;
    DWORD    dwAlphaOverlayConstBitDepths;
    DWORD    dwAlphaOverlayPixelBitDepths;
    DWORD    dwAlphaOverlaySurfaceBitDepths;
    DWORD    dwZBufferBitDepths;

    DWORD    dwVidMemTotal;
    DWORD    dwVidMemFree;
    DWORD    dwMaxVisibleOverlays;
    DWORD    dwCurrVisibleOverlays;
    DWORD    dwNumFourCCCodes;
    DWORD    dwAlignBoundarySrc;
    DWORD    dwAlignSizeSrc;
    DWORD    dwAlignBoundaryDest;
    DWORD    dwAlignSizeDest;
    DWORD    dwAlignStrideAlign;
    DWORD    dwRops[DD_ROP_SPACE];
    DDSCAPS  ddsCaps;
    DWORD    dwMinOverlayStretch;
    DWORD    dwMaxOverlayStretch;
    DWORD    dwMinLiveVideoStretch;

    DWORD    dwMaxLiveVideoStretch;
    DWORD    dwMinHwCodecStretch;
    DWORD    dwMaxHwCodecStretch;
    DWORD    dwReserved1;
    DWORD    dwReserved2;
    DWORD    dwReserved3;
    DWORD    dwSVBCaps;
    DWORD    dwSVBCKeyCaps;
    DWORD    dwSVBFXCaps;
    DWORD    dwSVBRops[DD_ROP_SPACE];
    DWORD    dwVSBCaps;
    DWORD    dwVSBCKeyCaps;
    DWORD    dwVSBFXCaps;
    DWORD    dwVSBRops[DD_ROP_SPACE];
    DWORD    dwSSBCaps;
    DWORD    dwSSBCKeyCaps;

    DWORD    dwSSBCFXCaps;
    DWORD    dwSSBRops[DD_ROP_SPACE];
    DWORD    dwReserved4;
    DWORD    dwReserved5;
    DWORD    dwReserved6;
} DDCAPS,FAR* LPDDCAPS;
```

Represents the capabilities of the hardware exposed through the DirectDraw object. It contains a **DDSCAPS** structure used in this context to describe what kinds of DirectDrawSurfaces can be created. It may not be possible to simultaneously create all of the surfaces described by these capabilities.

dwSize
Size of the structure. This must be initialized before the structure is used.

dwCaps
Specifies driver-specific capabilities.

DDCAPS_3D
Indicates the display hardware has 3D acceleration.

DDCAPS_ALIGNBOUNDARYDEST
Indicates that DirectDraw will support only those source rectangles with the x-axis aligned to the DDCAPS.dwAlignBoundaryDest boundaries of the surface.

DDCAPS_ALIGNBOUNDARYSRC
Indicates that DirectDraw will support only those source rectangles with the x-axis aligned to the DDCAPS.dwAlignBoundarySrc boundaries of the surface.

DDCAPS_ALIGNSIZEDEST
Indicates that DirectDraw will support only those source rectangles whose x-axis size, in bytes, are DDCAPS.dwAlignSizeDest multiples.

DDCAPS_ALIGNSIZESRC
Indicates that DirectDraw will support only those source rectangles whose x-axis size, in bytes, are DDCAPS.dwAlignSizeSrc multiples.

DDCAPS_ALIGNSTRIDE
Indicates that DirectDraw will create display memory surfaces that have a stride alignment equal to the DDCAPS.dwAlignStrideAlign value.

DDCAPS_ALPHA

Indicates the display hardware supports an alpha channel during blit operations.

DDCAPS_BANKSWITCHED

Indicates the display hardware is bank switched, and is potentially very slow at random access to VRAM.

DDCAPS_BLT

Indicates that display hardware is capable of blit operations.

DDCAPS_BLTCOLORFILL

Indicates that display hardware is capable of color filling with blitter.

DDCAPS_BLTDEPTHFILL

Indicates that display hardware is capable of depth filling z-buffers with blitter.

DDCAPS_BLTFOURCC

Indicates that display hardware is capable of color space conversions during blit operations.

DDCAPS_BLTQUEUE

Indicates that display hardware is capable of asynchronous blit operations.

DDCAPS_BLTSTRETCH

Indicates that display hardware is capable of stretching during blit operations.

DDCAPS_CANBLTSYSMEM

Indicates that display hardware is capable of blitting to or from system memory.

DDCAPS_CANCLIP

Indicates that display hardware is capable of clipping with blitting.

DDCAPS_CANCLIPSTRETCHED

Indicates that display hardware is capable of clipping while stretch blitting.

DDCAPS_COLORKEY

Supports some form of color key in either overlay or blit operations. More specific color key capability information can be found in the **dwCKeyCaps** member.

DDCAPS_COLORKEYHWASSIST

Indicates the color key is hardware assisted.

DDCAPS_GDI

Indicates that display hardware is shared with GDI.

DDCAPS_NOHARDWARE

Indicates no hardware support.

DDCAPS_OVERLAY

Indicates that display hardware supports overlays.

DDCAPS_OVERLAYCANTCLIP

Indicates that display hardware supports overlays but cannot clip them.

DDCAPS_OVERLAYFOURCC

Indicates that overlay hardware is capable of color space conversions during overlay operations.

DDCAPS_OVERLAYSTRETCH

Indicates that overlay hardware is capable of stretching.

DDCAPS_PALETTE

Indicates that DirectDraw is capable of creating and supporting DirectDrawPalette objects for more surfaces than only the primary surface.

DDCAPS_PALETTEVSYNC

Indicates that DirectDraw is capable of updating a palette synchronized with the vertical refresh.

DDCAPS_READSCANLINE

Indicates that display hardware is capable of returning the current scan line.

DDCAPS_STEREOVIEW

Indicates that display hardware has stereo vision capabilities.

DDCAPS_VBI

Indicates that display hardware is capable of generating a vertical blank interrupt.

DDCAPS_ZBLTS

Supports the use of z-buffers with blit operations.

DDCAPS_ZOVERLAYS

Supports the use of **IDirectDrawSurface::UpdateOverlayZOrder** as a z-value for overlays to control their layering.

dwCaps2

Specifies more driver-specific capabilities.

DDCAPS2_CERTIFIED

Indicates that display hardware is certified.

DDCAPS2_NO2DDURING3DSCENE

Indicates that 2D operations such as **IDirectDrawSurface::Blt** and **IDirectDrawSurface::Lock** cannot be performed on any surfaces that Direct3D is using between **IDirect3DDevice::BeginScene** and **IDirect3DDevice::EndScene** method calls.

dwCKeyCaps

Color key capabilities.

DDCKEYCAPS_DESTBLT

Supports transparent blitting with a color key that identifies the replaceable bits of the destination surface for RGB colors.

DDCKEYCAPS_DESTBLTCLRSPACE

Supports transparent blitting with a color space that identifies the replaceable bits of the destination surface for RGB colors.

DDCKEYCAPS_DESTBLTCLRSPACEYUV

Supports transparent blitting with a color space that identifies the replaceable bits of the destination surface for YUV colors.

DDCKEYCAPS_DESTBLTYUV

Supports transparent blitting with a color key that identifies the replaceable bits of the destination surface for YUV colors.

DDCKEYCAPS_DESTOVERLAY

Supports overlaying with color keying of the replaceable bits of the destination surface being overlaid for RGB colors.

DDCKEYCAPS_DESTOVERLAYCLRSPACE

Supports a color space as the color key for the destination of RGB colors.

DDCKEYCAPS_DESTOVERLAYCLRSPACEYUV

Supports a color space as the color key for the destination of YUV colors.

DDCKEYCAPS_DESTOVERLAYONEACTIVE

Supports only one active destination color key value for visible overlay surfaces.

DDCKEYCAPS_DESTOVERLAYYUV

Supports overlaying using color keying of the replaceable bits of the destination surface being overlaid for YUV colors.

DDCKEYCAPS_NOCOSTOVERLAY

Indicates there are no bandwidth trade-offs for using color key with an overlay.

DDCKEYCAPS_SRCBLT

Supports transparent blitting using the color key for the source with this surface for RGB colors.

DDCKEYCAPS_SRCBLTCLRSPACE

Supports transparent blitting using a color space for the source with this surface for RGB colors.

DDCKEYCAPS_SRCBLTCLRSPACEYUV

Supports transparent blitting using a color space for the source with this surface for YUV colors.

DDCKEYCAPS_SRCBLTYUV

Supports transparent blitting using the color key for the source with this surface for YUV colors.

DDCKEYCAPS_SRCOVERLAY

Supports overlaying using the color key for the source with this overlay surface for RGB colors.

DDCKEYCAPS_SRCOVERLAYCLRSPACE

Supports overlaying using a color space as the source color key for the overlay surface for RGB colors.

DDCKEYCAPS_SRCOVERLAYCLRSPACEYUV

Supports overlaying using a color space as the source color key for the overlay surface for YUV colors.

DDCKEYCAPS_SRCOVERLAYONEACTIVE

Supports only one active source color key value for visible overlay surfaces.

DDCKEYCAPS_SRCOVERLAYYUV

Supports overlaying using the color key for the source with this overlay surface for YUV colors.

dwFXCaps

Specifies driver-specific stretching and effects capabilities.

DDFXCAPS_BLTARITHSTRETCHY

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during a blit operation. Occurs along the y-axis (vertically).

DDFXCAPS_BLTARITHSTRETCHYN

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during a blit operation. Occurs along the y-axis (vertically), and only works for integer stretching (×1, ×2, and so on).

DDFXCAPS_BLTMIRRORLEFTRIGHT

Supports mirroring left to right in a blit operation.

DDFXCAPS_BLTMIRRORUPDOWN

Supports mirroring top to bottom in a blit operation.

DDFXCAPS_BLTROTATION

Supports arbitrary rotation in a blit operation.

DDFXCAPS_BLTROTATION90

Supports 90 degree rotations in a blit operation.

DDFXCAPS_BLTSHRINKX

Supports arbitrary shrinking of a surface along the x-axis (horizontally). This flag is only valid for blit operations.

DDFXCAPS_BLTSHRINKXN

Supports integer shrinking (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is only valid for blit operations.

DDFXCAPS_BLTSHRINKY

Supports arbitrary shrinking of a surface along the y-axis (vertically). This flag is only valid for blit operations.

DDFXCAPS_BLTSHRINKYN

Supports integer shrinking (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is only valid for blit operations.

DDFXCAPS_BLTSTRETCHX

Supports arbitrary stretching of a surface along the x-axis (horizontally). This flag is only valid for blit operations.

DDFXCAPS_BLTSTRETCHXN

Supports integer stretching (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is only valid for blit operations.

DDFXCAPS_BLTSTRETCHY

Supports arbitrary stretching of a surface along the y-axis (vertically). This flag is only valid for blit operations.

DDFXCAPS_BLTSTRETCHYN

Supports integer stretching (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is only valid for blit operations.

DDFXCAPS_OVERLAYARITHSTRETCHY

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during an overlay operation. Occurs along the y-axis (vertically).

DDFXCAPS_OVERLAYARITHSTRETCHYN

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during an overlay operation. Occurs along the y-axis (vertically), and only works for integer stretching (×1, ×2, and so on).

DDFXCAPS_OVERLAYMIRRORLEFTRIGHT

Supports mirroring of overlays around the vertical axis.

DDFXCAPS_OVERLAYMIRRORUPDOWN

Supports mirroring of overlays across the horizontal axis.

DDFXCAPS_OVERLAYSHRINKX

Supports arbitrary shrinking of a surface along the x-axis (horizontally). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKXN

Supports integer shrinking (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKY

Supports arbitrary shrinking of a surface along the y-axis (vertically). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKYN

Supports integer shrinking (x1, x2, and so on) of a surface along the y-axis (vertically). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSTRETCHX

Supports arbitrary stretching of a surface along the x-axis (horizontally). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHXN

Supports integer stretching (x1, x2, and so on) of a surface along the x-axis (horizontally). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHY

Supports arbitrary stretching of a surface along the y-axis (vertically). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHYN

Supports integer stretching (x1, x2, and so on) of a surface along the y-axis (vertically). This flag is only valid for DDSCAPS_OVERLAY surfaces. This flag only indicates the capabilities of a surface; it does not indicate that stretching is available.

dwFXAlphaCaps

Specifies driver-specific alpha capabilities.

DDFXALPHACAPS_BLTALPHAEDGEBLEND

Supports alpha blending around the edge of a source color keyed surface. Used for blit operations.

DDFXALPHACAPS_BLTALPHAPIXELS

Supports alpha information in pixel format. The bit depth of alpha information in the pixel format can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for blit operations.

DDFXALPHACAPS_BLTALPHAPIXELSNEG

Supports alpha information in pixel format. The bit depth of alpha information in the pixel format can be 1, 2, 4, or 8. The alpha value becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can only be set if DDCAPS_ALPHA is set. Used for blit operations.

DDFXALPHACAPS_BLTALPHASURFACES

Supports alpha-only surfaces. The bit depth of an alpha-only surface can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for blit operations.

DDFXALPHACAPS_BLTALPHASURFACESNEG

Indicates the depth of the alpha channel data can be 1, 2, 4, or 8. The NEG suffix indicates this alpha channel becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can only be set if DDFXCAPS_ALPHASURFACES has been set. Used for blit operations.

DDFXALPHACAPS_OVERLAYALPHAEDGEBLEND

Supports alpha blending around the edge of a source color keyed surface. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHAPIXELS

Supports alpha information in pixel format. The bit depth of alpha information in pixel format can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHAPIXELSNEG

Supports alpha information in pixel format. The bit depth of alpha information in pixel format can be 1, 2, 4, or 8. The alpha value becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can only be set if DDFXCAPS_ALPHAPIXELS has been set. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHASURFACES

Supports alpha-only surfaces. The bit depth of an alpha-only surface can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHASURFACESNEG

Indicates the depth of the alpha channel data can be 1, 2, 4, or 8. The NEG suffix indicates this alpha channel becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can only be set if DDFXCAPS_ALPHASURFACES has been set. Used for overlays.

dwPalCaps
Specifies palette capabilities.

DDPCAPS_1BIT
The index is 1 bit. There are two entries in the palette table.

DDPCAPS_2BIT
The index is 2 bits. There are four entries in the palette table.

DDPCAPS_4BIT
The index is 4 bits. There are sixteen entries in the palette table.

DDPCAPS_8BIT
The index is 8 bits. There are 256 entries in the palette table.

DDPCAPS_8BITENTRIES
An index to an 8-bit color index. This field is only valid when used with the DDPCAPS_1BIT, DDPCAPS_2BIT, or DDPCAPS_4BIT capability and when the target surface is in 8 bits per pixel (bpp). Each color entry is one byte long and is an index to an 8-bpp palette on the destination surface.

DDPCAPS_ALLOW256
Indicates this palette can have all 256 entries defined.

DDPCAPS_PRIMARYSURFACE
Indicates the palette is attached to the primary surface. Changing this table has an immediate effect on the display unless the DDPCAPS_VSYNC capability is specified and supported.

DDPCAPS_PRIMARYSURFACELEFT
Indicates the palette is attached to the primary surface on the left. Changing this table has an immediate effect on the display unless the DDPCAPS_VSYNC capability is specified and supported.

DDPCAPS_VSYNC
Indicates the palette can be modified synchronously with the monitor's refresh rate.

dwSVCaps
Specifies stereo vision capabilities.

DDSVCAPS_ENIGMA
Indicates the stereo view is accomplished using Enigma encoding.

DDSVCAPS_FLICKER
Indicates the stereo view is accomplished using high-frequency flickering.

DDSVCAPS_REDBLUE
Indicates the stereo view is accomplished using red and blue filters applied to the left and right eyes. All images must adapt their color spaces for this process.

DDSVCAPS_SPLIT

Indicates the stereo view is accomplished with split-screen technology.

dwAlphaBltConstBitDepths
DDBD_2,4,8

dwAlphaBltPixelBitDepths
DDBD_1,2,4,8

dwAlphaBltSurfaceBitDepths
DDBD_1,2,4,8

dwAlphaOverlayConstBitDepths
DDBD_2,4,8

dwAlphaOverlayPixelBitDepths
DDBD_1,2,4,8

dwAlphaOverlaySurfaceBitDepths
DDBD_1,2,4,8

dwZBufferBitDepths
DDBD_8,16,24,32

dwVidMemTotal
Total amount of display memory.

dwVidMemFree
Amount of free display memory.

dwMaxVisibleOverlays
Maximum number of visible overlays.

dwCurrVisibleOverlays
Current number of visible overlays.

dwNumFourCCCodes
Number of FourCC codes.

dwAlignBoundarySrc
Source rectangle alignment.

dwAlignSizeSrc
Source rectangle byte size.

dwAlignBoundaryDest
Destination rectangle alignment.

dwAlignSizeDest
Destination rectangle byte size.

dwAlignStrideAlign
Stride alignment.

dwRops[DD_ROP_SPACE]
Raster operations supported.

ddsCaps
Indicates a **DDSCAPS** structure with general capabilities.

dwMinOverlayStretch
Minimum overlay stretch factor multiplied by 1000. For example, 1.0 = 1000, 1.3 = 1300.

dwMaxOverlayStretch
Maximum overlay stretch factor multiplied by 1000. For example, 1.0 = 1000, 1.3 = 1300.

dwMinLiveVideoStretch
Minimum live video stretch factor multiplied by 1000. For example, 1.0 = 1000, 1.3 = 1300.

dwMaxLiveVideoStretch
Maximum live video stretch factor multiplied by 1000. For example, 1.0 = 1000, 1.3 = 1300.

dwMinHwCodecStretch
Minimum hardware codec stretch factor multiplied by 1000. For example 1.0 = 1000, 1.3 = 1300.

dwMaxHwCodecStretch
Maximum hardware codec stretch factor multiplied by 1000. For example, 1.0 = 1000, 1.3 = 1300.

dwReserved1,**dwReserved2**,**dwReserved3**
Reserved for future use.

dwSVBCaps
Specifies the driver-specific capabilities for system memory to display memory blits.

dwSVBCKeyCaps
Specifies the driver color key capabilities for system memory to display memory blits.

dwSVBFXCaps
Specifies the driver FX capabilities for system memory to display memory blits.

dwSVBRops[DD_ROP_SPACE]
Specifies the raster operations supported for system memory to display memory blits.

dwVSBCaps
Specifies the driver-specific capabilities for display memory to system memory blits.

dwVSBCKeyCaps
Specifies the driver color key capabilities for display memory to system memory blits.

dwVSBFXCaps
Specifies the driver FX capabilities for display memory to system memory blits.

dwVSBRops[DD_ROP_SPACE]
Supports raster operations for display memory to system memory blits.

dwSSBCaps
Specifies the driver-specific capabilities for system memory to system memory blits.

dwSSBCKeyCaps
Specifies the driver color key capabilities for system memory to system memory blits.

dwSSBCFXCaps
Specifies the driver FX capabilities for system memory to system memory blits.

dwSSBRops[DD_ROP_SPACE]
Raster operations supported for system memory to system memory blits.

dwReserved4, **dwReserved5**, **dwReserved6**
Reserved for future use.

The values for the **dw...BitDepths** members (**dwAlphaBltConstBitDepths**, **dwAlphaBltPixelBitDepths**, **dwAlphBltSurfaceBitDepths**, and so on) are:

DDBD_1	1 bit per pixel.
DDBD_2	2 bits per pixel.
DDBD_4	4 bits per pixel.
DDBD_8	8 bits per pixel.
DDBD_16	16 bits per pixel.
DDBD_24	24 bits per pixel.
DDBD_32	32 bits per pixel.

DDCOLORKEY

```
typedef struct _DDCOLORKEY{
    DWORD  dwColorSpaceLowValue;
    DWORD  dwColorSpaceHighValue;
} DDCOLORKEY,FAR* LPDDCOLORKEY;
```

Describes a source or destination color key or color space. A color key is specified if the low and high range values are the same.

dwColorSpaceLowValue
Low value, inclusive, of the color range that is to be used as the color key.

dwColorSpaceHighValue
High value, inclusive, of the color range that is to be used as the color key.

DDOVERLAYFX

```
typedef struct _DDOVERLAYFX{
    DWORD  dwSize;
    DWORD  dwAlphaEdgeBlendBitDepth;
    DWORD  dwAlphaEdgeBlend;
    DWORD  dwReserved;
    DWORD  dwAlphaDestConstBitDepth;
union
{
        DWORD                dwAlphaDestConst;
        LPDIRECTDRAWSURFACE  lpDDSAlphaDest;
};
    DWORD  dwAlphaSrcConstBitDepth;
union
{
        DWORD                dwAlphaSrcConst;
        LPDIRECTDRAWSURFACE  lpDDSAlphaSrc;
};
    DDCOLORKEY  dckDestColorkey;
    DDCOLORKEY  dckSrcColorkey;

    DWORD       dwDDFX;
    DWORD       dwFlags;
} DDOVERLAYFX,FAR *LPDDOVERLAYFX;
```

Passes override information to the **IDirectDrawSurface::UpdateOverlay** method.

dwSize
Size of the structure. This must be initialized before the structure is used.

dwAlphaEdgeBlendBitDepth
Bit depth used to specify the constant for an alpha edge blend.

dwAlphaEdgeBlend
Constant to use as the alpha for an edge blend.

dwReserved
Reserved for future use.

dwAlphaDestConstBitDepth
Bit depth used to specify the alpha constant for a destination.

dwAlphaDestConst
Constant to use as the alpha channel for a destination.

lpDDSAlphaDest
Address of a surface to use as the alpha channel for a destination.

dwAlphaSrcConstBitDepth
Bit depth used to specify the alpha constant for a source.

dwAlphaSrcConst
Constant to use as the alpha channel for a source.

lpDDSAlphaSrc
Address of a surface to use as the alpha channel for a source.

dckDestColorkey
Destination color key override.

dckSrcColorkey
Source color key override.

dwDDFX
Overlay FX flags.I

DDOVERFX_ARITHSTRETCHY
If stretching, use arithmetic stretching along the y-axis for this overlay.

DDOVERFX_MIRRORLEFTRIGHT
Mirror the overlay around the vertical axis.

DDOVERFX_MIRRORUPDOWN
Mirror the overlay around the horizontal axis.

dwFlags
This member is not used at this time and must be set to 0.

DDPIXELFORMAT

```
typedef struct _DDPIXELFORMAT{
    DWORD  dwSize;
    DWORD  dwFlags;
    DWORD  dwFourCC;
union
{
        DWORD  dwRGBBitCount;
        DWORD  dwYUVBitCount;
        DWORD  dwZBufferBitDepth;
        DWORD  dwAlphaBitDepth;
};
union
{
        DWORD  dwRBitMask;
        DWORD  dwYBitMask;
};
union
{
        DWORD  dwGBitMask;
        DWORD  dwUBitMask;
};
union
{
        DWORD  dwBBitMask;
        DWORD  dwVBitMask;
};
union
{
        DWORD  dwRGBAlphaBitMask;

        DWORD  dwYUVAlphaBitMask;
};
} DDPIXELFORMAT, FAR* LPDDPIXELFORMAT;
```

Describes the pixel format of a DirectDrawSurface object.

dwSize
Size of the structure. This must be initialized before the structure is used.

dwFlags
Specifies the optional control flags.

DDPF_ALPHA
The pixel format describes an alpha-only surface.

DDPF_ALPHAPIXELS
The surface has alpha-channel information in the pixel format.

DDPF_COMPRESSED
The surface will accept pixel data in the specified format and compress it during the write operation.

DDPF_FOURCC
The FourCC code is valid.

DDPF_PALETTEINDEXED1
The surface is 1-bit color indexed.

DDPF_PALETTEINDEXED2
The surface is 2-bit color indexed.

DDPF_PALETTEINDEXED4
The surface is 4-bit color indexed.

DDPF_PALETTEINDEXED8
The surface is 8-bit color indexed.

DDPF_PALETTEINDEXEDTO8
The surface is 1-, 2-, or 4-bit color indexed to an 8-bit palette.

DDPF_RGB
The RGB data in the pixel format structure is valid.

DDPF_RGBTOYUV
The surface will accept RGB data and translate it during the write operation to YUV data. The format of the data to be written will be contained in the pixel format structure. The DDPF_RGB flag will be set.

DDPF_YUV

The YUV data in the pixel format structure is valid.

DDPF_ZBUFFER

The pixel format describes a z-buffer-only surface.

dwFourCC
FourCC code.

dwRGBBitCount
RGB bits per pixel (DDBD_4,8,16,24,32)

dwYUVBitCount
YUV bits per pixel (DDBD_4,8,16,24,32)

dwZBufferBitDepth
Z-buffer bit depth. (DDBD_8,16,24,32)

dwAlphaBitDepth
Alpha channel bit depth. (DDBD_1,2,4,8)

dwRBitMask
Mask for red bits.

dwYBitMask
Mask for Y bits.

dwGBitMask
Mask for green bits.

dwUBitMask
Mask for U bits.

dwBBitMask
Mask for blue bits.

dwVBitMask
Mask for V bits.

dwRGBAlphaBitMask
Mask for alpha channel.

dwYUVAlphaBitMask
Mask for alpha channel.

DDSCAPS

```
typedef struct _DDSCAPS{
        DWORD  dwCaps;
} DDSCAPS,  FAR* LPDDSCAPS;
```

Defines the capabilities of a DirectDrawSurface. It is part of the **DDCAPS** structure that is used to describe the capabilities of the DirectDraw object.

dwCaps
Specifies the capabilities of the surface.

DDSCAPS_3D

Supported for backwards compatibility. Applications should use the DDSCAPS_3DDEVICE flag, instead.

DDSCAPS_3DDEVICE

Indicates that this surface can be used for 3D rendering. Applications can use this flag to ensure that a device that can only render to a certain heap have off-screen surfaces allocated from the correct heap. If this flag is set for a heap, the surface will not be allocated from that heap.

DDSCAPS_ALLOCONLOAD

Indicates that memory for the surface is not allocated until the surface is loaded using the **IDirect3DTexture::Load** method.

DDSCAPS_ALPHA

Indicates that this surface contains alpha information. The pixel format must be interrogated to determine whether this surface contains only alpha information or alpha information interlaced with pixel color data (such as RGBA or YUVA).

DDSCAPS_BACKBUFFER

Indicates that this surface is THE back buffer of a surface flipping structure. Generally, this capability is set by the **IDirectDraw::CreateSurface** method when the DDSCAPS_FLIP capability is set. Only the surface that immediately precedes the DDSCAPS_FRONTBUFFER has this capability set. The other surfaces are identified as back buffers by the presence of the DDSCAPS_FLIP flag, their attachment order, and the absence of the DDSCAPS_FRONTBUFFER and DDSCAPS_BACKBUFFER capabilities. If this capability is sent to the **IDirectDraw::CreateSurface** method, a standalone back buffer is being created. This surface could be attached to a front buffer, another back buffer, or both to form a flipping surface structure after this method is called. See the **IDirectDrawSurface::AddAttachedSurface** method for a detailed description of the behaviors in this case. DirectDraw supports *n* surfaces in a surface flipping structure.

DDSCAPS_COMPLEX

Indicates a complex surface structure is being described. A complex surface structure results in the creation of more than one surface. The additional surfaces are attached to the root surface. The complex structure can only be destroyed by destroying the root.

DDSCAPS_FLIP

Indicates that this surface is a part of a surface flipping structure. When this capability is passed to the **IDirectDraw::CreateSurface** method, the DDSCAPS_FRONTBUFFER and DDSCAPS_BACKBUFFER capabilities are not set. Both can be set by this method on resulting creations. The **dwBackBufferCount** member in the **DDSURFACEDESC** structure must be set to at least 1 in order for the method call to succeed. The DDSCAPS_COMPLEX capability must always be set when creating multiple surfaces through the **IDirectDraw::CreateSurface** method.

DDSCAPS_FRONTBUFFER

Indicates that this surface is the front buffer of a surface flipping structure. It is generally set by the **IDirectDraw::CreateSurface** method when the DDSCAPS_FLIP capability is set. If this capability is sent to the **IDirectDraw::CreateSurface** method, a standalone front buffer is created. This surface will not have the DDSCAPS_FLIP capability. It can be attached to other back buffers to form a flipping structure on resulting creations. See the **IDirectDrawSurface::AddAttachedSurface** method for a detailed description of the behaviors in this case.

DDSCAPS_HWCODEC

Indicates that this surface should be able to have a stream decompressed to it by the hardware.

DDSCAPS_LIVEVIDEO

Indicates that this surface should be able to receive live video.

DDSCAPS_MIPMAP

Indicates that this surface is one level of a mipmap. This surface will be attached to other DDSCAPS_MIPMAP surfaces to form the mipmap. This can be done explicitly by creating a number of surfaces and attaching them with the **IDirectDrawSurface::AddAttachedSurface** method, or implicitly by the **IDirectDraw::CreateSurface** method. If this capability is set, then DDSCAPS_TEXTURE must also be set.

DDSCAPS_MODEX

Indicates that this surface is a 320 x 200 or 320 x 240 ModeX surface.

DDSCAPS_OFFSCREENPLAIN

Indicates that this surface is any off-screen surface that is not an overlay, texture, z-buffer, front buffer, back buffer, or alpha surface. It is used to identify plain surfaces.

DDSCAPS_OVERLAY

Indicates that this surface is an overlay. It may or may not be directly visible depending on whether it is currently being overlaid onto the primary surface. DDSCAPS_VISIBLE can be used to determine if it is being overlaid at the moment.

DDSCAPS_OWNDC

Indicates that this surface will have a DC association for a long period.

DDSCAPS_PALETTE

Indicates that this device driver allows unique DirectDrawPalette objects to be created and attached to this surface.

DDSCAPS_PRIMARYSURFACE

Indicates the surface is the primary surface. It represents what is visible to the user at the moment.

DDSCAPS_PRIMARYSURFACELEFT

Indicates that this surface is the primary surface for the left eye. It represents what is visible to the user's left eye at the moment. When this surface is created, the surface with the DDSCAPS_PRIMARYSURFACE capability represents what is seen by the user's right eye.

DDSCAPS_SYSTEMMEMORY

Indicates that this surface memory was allocated in system memory.

DDSCAPS_TEXTURE

Indicates that this surface can be used as a 3D texture. It does not indicate if the surface is being used for that purpose.

DDSCAPS_VIDEOMEMORY

Indicates that this surface exists in display memory.

DDSCAPS_VISIBLE

Indicates that changes made to this surface are immediately visible. It is always set for the primary surface and is set for overlays while they are being overlaid and texture maps while they are being textured.

DDSCAPS_WRITEONLY

Indicates that only write access is permitted to the surface. Read access from the surface may generate a protection fault, but the read results from this surface will not be meaningful.

DDSCAPS_ZBUFFER

Indicates that this surface is the z-buffer. The z-buffer contains information that cannot be displayed. Instead, it contains bit-depth information that is used to determine which pixels are visible and which are obscured.

DDSURFACEDESC

```
typedef struct _DDSURFACEDESC{
    DWORD  dwSize;
    DWORD  dwFlags;
    DWORD  dwHeight;
    DWORD  dwWidth;
    LONG   lPitch,
    DWORD  dwBackBufferCount,
    union
    {
        DWORD  dwMipMapCount,
        DWORD  dwZBufferBitDepth,
        DWORD  dwRefreshRate,
    };

    DWORD          dwAlphaBitDepth;
    DWORD          dwReserved;
    LPVOID          lpSurface;
    DDCOLORKEY      ddckCKDestOverlay;
    DDCOLORKEY      ddckCKDestBlt;

    DDCOLORKEY      ddckCKSrcOverlay;
    DDCOLORKEY      ddckCKSrcBlt;
    DDPIXELFORMAT   ddpfPixelFormat;
    DDSCAPS         ddsCaps;
} DDSURFACEDESC,  FAR* LPDDSURFACEDESC;
```

Is passed to the **IDirectDraw::CreateSurface** method with a description of the surface that should be created. The relevant members differ for each potential type of surface.

dwSize
Size of the structure. This must be initialized before the structure is used.

dwFlags
Specifies the optional control flags.

DDSD_ALL
All input members are valid.

DDSD_ALPHABITDEPTH
Indicates the **dwAlphaBitDepth** member is valid.

DDSD_BACKBUFFERCOUNT
Indicates the **dwBackBufferCount** member is valid.

DDSD_CAPS
Indicates the **ddsCaps** member is valid.

DDSD_CKDESTBLT
Indicates the **ddckCKDestBlt** member is valid.

DDSD_CKDESTOVERLAY
Indicates the **ddckCKDestOverlay** member is valid.

DDSD_CKSRCBLT
Indicates the **ddckCKSrcBlt** member is valid.

DDSD_CKSRCOVERLAY
Indicates the **ddckCKSrcOverlay** member is valid.

DDSD_HEIGHT
Indicates the **dwHeight** member is valid.

DDSD_LPSURFACE
Indicates the **lpSurface** member is valid.

DDSD_MIPMAPCOUNT
Indicates the **dwMipMapCount** member is valid.

DDSD_PITCH

Indicates the **lPitch** member is valid.

DDSD_PIXELFORMAT

Indicates the **ddpfPixelFormat** member is valid.

DDSD_REFRESHRATE

Indicates the **dwRefreshRate** member is valid.

DDSD_WIDTH

Indicates the **dwWidth** member is valid.

DDSD_ZBUFFERBITDEPTH

Indicates the **dwZBufferBitDepth** member is valid.

dwHeight
Height of surface.

dwWidth
Width of input surface.

lPitch
Distance to start of next line (return value only).

dwBackBufferCount
Number of back buffers.

dwMipMapCount
Number of mipmap levels.

dwZBufferBitDepth
Depth of z-buffer.

dwRefreshRate
Refresh rate (used when the display mode is described).

dwAlphaBitDepth
Depth of alpha buffer.

dwReserved
Reserved.

lpSurface
Address of the associated surface memory.

ddckCKDestOverlay
Color key for destination overlay use.

ddckCKDestBlt
Color key for destination blit use.

ddckCKSrcOverlay
Color key for source overlay use.

ddckCKSrcBlt
Color key for source blit use.

ddpfPixelFormat
Pixel format description of the surface.

ddsCaps
DirectDraw surface capabilities.

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all IDirectDraw, IDirectDrawSurface, IDirectDrawPalette, and IDirectDrawClipper methods. For a list of the error codes each method is capable of returning, see the individual method descriptions.

DD_OK
The request completed successfully.

DDERR_ALREADYINITIALIZED
The object has already been initialized.

DDERR_BLTFASTCANTCLIP
A clipper object is attached to a source surface that has passed into a call to the **IDirectDrawSurface::BltFast** method.

DDERR_CANNOTATTACHSURFACE
A surface cannot be attached to another requested surface.

DDERR_CANNOTDETACHSURFACE
A surface cannot be detached from another requested surface.

DDERR_CANTCREATEDC
Windows cannot create any more device contexts (DCs).

DDERR_CANTDUPLICATE
Primary and 3D surfaces, or surfaces that are implicitly created, cannot be duplicated.

DDERR_CANTLOCKSURFACE
Access to this surface is refused because an attempt was made to lock the primary surface without DCI support.

DDERR_CANTPAGELOCK
An attempt to page lock a surface failed. Page lock will not work on a display memory surface or an emulated primary surface.

DDERR_CANTPAGEUNLOCK
An attempt to page unlock a surface failed. Page unlock will not work on a display memory surface or an emulated primary surface.

DDERR_CLIPPERISUSINGHWND
An attempt was made to set a clip list for a clipper object that is already monitoring an hWnd.

DDERR_COLORKEYNOTSET
No source color key is specified for this operation.

DDERR_CURRENTLYNOTAVAIL
No support is currently available.

DDERR_DCALREADYCREATED
A device context has already been returned for this surface. Only one device context can be retrieved for each surface.

DDERR_DIRECTDRAWALREADYCREATED
A DirectDraw object representing this driver has already been created for this process.

DDERR_EXCEPTION
An exception was encountered while performing the requested operation.

DDERR_EXCLUSIVEMODEALREADYSET
An attempt was made to set the cooperative level when it was already set to exclusive.

DDERR_GENERIC
There is an undefined error condition.

DDERR_HEIGHTALIGN
The height of the provided rectangle is not a multiple of the required alignment.

DDERR_HWNDALREADYSET
The DirectDraw CooperativeLevel HWND has already been set. It cannot be reset while the process has surfaces or palettes created.

DDERR_HWNDSUBCLASSED
DirectDraw is prevented from restoring state because the DirectDraw CooperativeLevel HWND has been subclassed.

DDERR_IMPLICITLYCREATED
The surface cannot be restored because it is an implicitly created surface.

DDERR_INCOMPATIBLEPRIMARY
The primary surface creation request does not match with the existing primary surface.

DDERR_INVALIDCAPS
One or more of the capability bits passed to the callback function are incorrect.

DDERR_INVALIDCLIPLIST
DirectDraw does not support the provided clip list.

DDERR_INVALIDDIRECTDRAWGUID
The GUID passed to the **DirectDrawCreate** function is not a valid DirectDraw driver identifier.

DDERR_INVALIDMODE

DirectDraw does not support the requested mode.

DDERR_INVALIDOBJECT

DirectDraw received a pointer that was an invalid DirectDraw object.

DDERR_INVALIDPARAMS

One or more of the parameters passed to the method are incorrect.

DDERR_INVALIDPIXELFORMAT

The pixel format was invalid as specified.

DDERR_INVALIDPOSITION

The position of the overlay on the destination is no longer legal.

DDERR_INVALIDRECT

The provided rectangle was invalid.

DDERR_INVALIDSURFACETYPE

The requested operation could not be performed because the surface was of the wrong type.

DDERR_LOCKEDSURFACES

One or more surfaces are locked, causing the failure of the requested operation.

DDERR_NO3D

No 3D is present.

DDERR_NOALPHAHW

No alpha acceleration hardware is present or available, causing the failure of the requested operation.

DDERR_NOBLTHW

No blitter hardware is present.

DDERR_NOCLIPLIST

No clip list is available.

DDERR_NOCLIPPERATTACHED

No clipper object is attached to the surface object.

DDERR_NOCOLORCONVHW

The operation cannot be carried out because no color conversion hardware is present or available.

DDERR_NOCOLORKEY

The surface does not currently have a color key.

DDERR_NOCOLORKEYHW

The operation cannot be carried out because there is no hardware support for the destination color key.

DDERR_NOCOOPERATIVELEVELSET

A create function is called without the **IDirectDraw::SetCooperativeLevel** method being called.

DDERR_NODC

No device context (DC) has ever been created for this surface.

DDERR_NODDROPSHW

No DirectDraw raster operation (ROP) hardware is available.

DDERR_NODIRECTDRAWHW

Hardware-only DirectDraw object creation is not possible; the driver does not support any hardware.

DDERR_NODIRECTDRAWSUPPORT

DirectDraw support is not possible with the current display driver.

DDERR_NOEMULATION

Software emulation is not available.

DDERR_NOEXCLUSIVEMODE

The operation requires the application to have exclusive mode, but the application does not have exclusive mode.

DDERR_NOFLIPHW

Flipping visible surfaces is not supported.

DDERR_NOGDI

No GDI is present.

DDERR_NOHWND

Clipper notification requires an HWND, or no HWND has been previously set as the CooperativeLevel HWND.

DDERR_NOMIPMAPHW

The operation cannot be carried out because no mipmap texture mapping hardware is present or available.

DDERR_NOMIRRORHW

The operation cannot be carried out because no mirroring hardware is present or available.

DDERR_NOOVERLAYDEST

The **IDirectDrawSurface::GetOverlayPosition** method is called on an overlay that the **IDirectDrawSurface::UpdateOverlay** method has not been called on to establish a destination.

DDERR_NOOVERLAYHW

The operation cannot be carried out because no overlay hardware is present or available.

DDERR_NOPALETTEATTACHED

No palette object is attached to this surface.

DDERR_NOPALETTEHW

There is no hardware support for 16- or 256-color palettes.

DDERR_NORASTEROPHW

The operation cannot be carried out because no appropriate raster operation hardware is present or available.

DDERR_NOROTATIONHW

The operation cannot be carried out because no rotation hardware is present or available.

DDERR_NOSTRETCHHW

The operation cannot be carried out because there is no hardware support for stretching.

DDERR_NOT4BITCOLOR

The DirectDrawSurface is not using a 4-bit color palette and the requested operation requires a 4-bit color palette.

DDERR_NOT4BITCOLORINDEX

The DirectDrawSurface is not using a 4-bit color index palette and the requested operation requires a 4-bit color index palette.

DDERR_NOT8BITCOLOR

The DirectDrawSurface is not using an 8-bit color palette and the requested operation requires an 8-bit color palette.

DDERR_NOTAOVERLAYSURFACE

An overlay component is called for a non-overlay surface

DDERR_NOTEXTUREHW

The operation cannot be carried out because no texture mapping hardware is present or available.

DDERR_NOTFLIPPABLE

An attempt has been made to flip a surface that cannot be flipped.

DDERR_NOTFOUND

The requested item was not found.

DDERR_NOTINITIALIZED

An attempt was made to invoke an interface method of a DirectDraw object created by **CoCreateInstance** before the object was initialized.

DDERR_NOTLOCKED

An attempt is made to unlock a surface that was not locked.

DDERR_NOTPAGELOCKED

An attempt is made to page unlock a surface with no outstanding page locks.

DDERR_NOTPALETTIZED

The surface being used is not a palette-based surface.

DDERR_NOVSYNCHW

The operation cannot be carried out because there is no hardware support for vertical blank synchronized operations.

DDERR_NOZBUFFERHW

The operation cannot be carried out because there is no hardware support for z-buffers when creating a z-buffer in display memory or when performing a z-aware blit.

DDERR_NOZOVERLAYHW

The overlay surfaces cannot be z-layered based on their *BltOrder* because the hardware does not support z-layering of overlays.

DDERR_OUTOFCAPS

The hardware needed for the requested operation has already been allocated.

DDERR_OUTOFMEMORY

DirectDraw does not have enough memory to perform the operation.

DDERR_OUTOFVIDEOMEMORY

DirectDraw does not have enough display memory to perform the operation.

DDERR_OVERLAYCANTCLIP

The hardware does not support clipped overlays.

DDERR_OVERLAYCOLORKEYONLYONEACTIVE

An attempt was made to have more than one color key active on an overlay.

DDERR_OVERLAYNOTVISIBLE

The **IDirectDrawSurface::GetOverlayPosition** method is called on a hidden overlay.

DDERR_PALETTEBUSY

Access to this palette is refused because the palette is locked by another thread.

DDERR_PRIMARYSURFACEALREADYEXISTS

This process has already created a primary surface.

DDERR_REGIONTOOSMALL

The region passed to the **IDirectDrawClipper::GetClipList** method is too small.

DDERR_SURFACEALREADYATTACHED

An attempt was made to attach a surface to another surface to which it is already attached.

DDERR_SURFACEALREADYDEPENDENT

An attempt was made to make a surface a dependency of another surface to which it is already dependent.

DDERR_SURFACEBUSY

Access to this surface is refused because the surface is locked by another thread.

DDERR_SURFACEISOBSCURED

Access to the surface is refused because the surface is obscured.

DDERR_SURFACELOST

Access to this surface is refused because the surface memory is gone. The DirectDrawSurface object representing this surface should have the **IDirectDrawSurface::Restore** method called on it.

DDERR_SURFACENOTATTACHED

The requested surface is not attached.

DDERR_TOOBIGHEIGHT

The height requested by DirectDraw is too large.

DDERR_TOOBIGSIZE

The size requested by DirectDraw is too large. However, the individual height and width are OK.

DDERR_TOOBIGWIDTH

The width requested by DirectDraw is too large.

DDERR_UNSUPPORTED

The operation is not supported.

DDERR_UNSUPPORTEDFORMAT

The FourCC format requested is not supported by DirectDraw.

DDERR_UNSUPPORTEDMASK

The bitmask in the pixel format requested is not supported by DirectDraw.

DDERR_UNSUPPORTEDMODE

The display is currently in an unsupported mode.

DDERR_VERTICALBLANKINPROGRESS

A vertical blank is in progress.

DDERR_WASSTILLDRAWING

The previous blit operation that is transferring information to or from this surface is incomplete.

DDERR_WRONGMODE

This surface cannot be restored because it was created in a different mode.

DDERR_XALIGN

The provided rectangle was not horizontally aligned on a required boundary.