Table Of Contents

Members

About DirectDraw

The Windows 95 Game SDK enables the creation of world class computer games. DirectDraw is a component of that SDK that allows direct manipulation of video display memory, hardware blters, hardware overlays, and page flipping. DirectDraw provides this functionality while maintaining compatibility with existing Windows 95 applications and device drivers.

The Windows Game Subsystem allows game authors an unprecedented level of access to the display and audio hardware while insulating them from the specific details of that hardware. *The Windows Game Subsystem is built for speed.* In keeping with these design goals, DirectDraw is not a high level graphics API.

DirectDraw for the Microsoft® Windows™ operating system is a software interface which provides direct access to display devices while maintaining compatibility with Windows GDI. It works with Windows 95 and will work with Windows NT. DirectDraw provides a device-independent way for games and Windows subsystem software such as 3-D graphics packages or digital video codecs to access display device-dependent features.

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

DirectDraw provides access to the following display device-dependent benefits:

- Support for double-buffered and page flipping graphics

- Access to, and control of, the video card's blitter
- Support for 3D z buffers
- Hardware assisted overlays with z ordering
- Improved graphics quality through access to image-stretching hardware
- Simultaneous access to standard and enhanced display device memory areas

DirectDraw provides world-class game graphics on a Windows 95 class PC. DirectDraw's mission is to provide device dependent access to video memory in a device independent way. The application need only pay attention to some basic device dependencies that are remarkably standard across hardware implementations, such as RGB and YUV color formats and the stride between raster lines. The application need not worry about the specific calling procedures required to utilize the blitter or manipulate palette registers. Essentially, DirectDraw is a memory manager for video memory. Using DirectDraw, an application can manipulate video memory with ease, taking full advantage of the blitting and color decompression capabilities of different types of video hardware without becoming dependent on a particular piece of hardware.

DirectDraw Overview

DirectDraw provides display memory and display hardware management services. DirectDraw 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. The DirectDraw HAL also exposes unique display hardware functionality, including stretching, overlaying, texture mapping, rotating and mirroring.

Architectural Overview

DirectDraw
DirectDraw HAL
DirectDraw Software Emulation
DirectDraw, DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper

DirectDraw

DirectDraw is the Windows system component that performs the common functions required by both hardware and software implementations of DirectDraw. DirectDraw is the only client of the DirectDraw HAL. Applications must write to DirectDraw. DirectDraw returns two sets of capabilities, one for hardware capabilities and one for software emulation capabilities. Using these, the 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 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. It does complete parameter validation. It provides the memory manager for offscreen video memory and performs all of the bookkeeping and semantic logic required for DirectDraw. It is responsible for presenting the COM interface to the application, hooking hWnd's 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-bit on Windows NT. The HAL may 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 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 itself to DirectDraw just like a HAL would. It has capabilities that it reports to DirectDraw, just like a HAL would. By examining these capabilities during application initialization, developers can adjust gaming 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 that functionality.

DirectDraw, DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper

The DirectDraw object represents the display device. There can be one DirectDraw object for every logical display device in operation. A game development environment, for instance, might have two monitors, one running the game 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 addresses may point to visible frame buffer memory (primary surface) or to non-visible buffers (offscreen or overlay surfaces). These non-visible buffers usually reside in video 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 warped onto a 3D surface described by the 3D DDI.

DirectDrawPalettes represent a 16 or 256 color-indexed palette. Palettes are provided for textures, offscreen surfaces, and overlay surfaces, all of which do not necessarily have the same palette as the primary surface.

The DirectDraw object creates DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper objects. DirectDrawPalettes and DirectDrawClippers must be attached to the DirectDrawSurface objects they affect. A DirectDrawSurface may refuse the request to attach a DirectDrawPalette to it. In practice, this is the usual behavior, since most hardware does not support multiple palettes.

Using DirectDraw

Frame buffer access

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

Experience has shown that developers run into several common problems when rendering directly into DirectDrawSurfaces: 1) Never assume a constant display pitch. Always examine the pitch information returned by the **Lock** member. This pitch may vary for a number of reasons, including the location of the surface memory, the type of video card, or even the version of the DirectDraw driver being used. 2) Limit activity between Lock and Unlock. The Lock member holds the WIN16 lock to safely access surface memory, and GetDC implicitly calls Lock. The WIN16 lock serializes access to GDI and USER, shutting down Windows between Lock/Unlock and GetDC/ReleaseDC pairs. 3) Copy aligned to video memory. Window 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 video memory can cause the system to hang if the copy happens to span memory banks.

Primary Surface resource sharing model

DirectDraw has a simple resource sharing model. Video memory is a scarce, shared resource. If the mode is changed, all of the surfaces stored in video memory are lost. (See **Losing surfaces,** below.)

DirectDraw implicitly creates a GDIPrimarySurface when it is instantiated for a display device it is sharing with GDI. GDI is granted shared access to the primary surface. DirectDraw will keep track of the surface memory that GDI believes is the primary surface. The DirectDrawSurface that owns GDI's primary surface can always be obtained using the **GetGDISurface** member of the DirectDraw object.

GDI is not allowed to cache fonts, brushes, and DDBs in the video memory managed by DirectDraw. The HAL must reserve whatever video memory the DIBENG driver needs before describing the available memory to DirectDraw's heap manager or 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 **SetDisplayMode** member of the DirectDraw object. Modes can be changed by any application as long as all of the applications are sharing the display card. If an application has obtained exclusive access to the display card, no other application can change the mode.

Exclusive mode in the context of DirectDraw does not mean that other applications can not allocate DirectDrawSurfaces or use DirectDraw functionality. It certainly does not mean that other applications cannot use GDI functionality. It does prevent applications other than the application which obtained exclusive access from changing the display mode or changing the palette.

Losing surfaces

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

Surfaces can be lost because the mode of the display card was changed or because an application received exclusive access to the display card and freed all of the surface memory currently allocated on the video card. The **Restore** member is provided to recreate these lost surfaces and reconnect them to their DirectDrawSurface objects.

Cliplists

Cliplists 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's clip list with the clip list for the window as it changes.

Although the clip list is visible from the DirectDraw HAL. The HAL need not be concerned about the clip list for blitting, as DirectDraw will only call the HAL with rectangles that meet the cliplist requirements. For instance, if the upper right rectangle of a surface was clipped and the application asked DirectDraw to blit the surface onto the primary surface DirectDraw would tell the HAL to 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 must consider the cliplist for overlays if the overlay hardware can support clipping and 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. In this case the HAL does not have to be concerned with clip lists, either.

Color and Format Conversion

The 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 field 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 will be valid masks which 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 will be valid masks which can be used to extract information from the pixels. The DDPF_RGB flag may be set in conjunction with the DDPF_FOURCC flag if a non-standard RGB format is being described.

There are two sets of FourCC codes exposed to the application. One list of FourCC codes represents what the blitting hardware is capable of doing and the other represents what the overlay hardware is capable of doing.

Specifying Color Keys

Color keys are specified in the pixel format of the surface. If the surface is in a palettized format, then the color key is specified as an index or a range of indexes. 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 each the dwColorSpaceLowValue and dwColorSpaceHighValue of the DDCOLORKEY structure. The lowest order byte has the V data, the second lowest order byte has the U data, and the third highest order byte has the Y data. The **SetColorKey** member has a "flags" parameter that specifies whether the color key being set 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 palletized 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);
```

Colorkeying

Both 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 is used to specify a color, or color range, that will not be copied in the case of blitting, or visible in the case of overlays, on the destination. Destination color keying is used to specify a color, or color range, that will be replaced in the case of blitting, or covered up, in the case of overlays, 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, then only the pixels that match the color key will be changed, or covered up, on the destination.

Some hardware will only support color ranges for YUV pixel data. This is because YUV data is usually video, and due to quantization 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.

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 a **Flip** 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 a **Flip** 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 **Lock** or **Blt** that target the back buffer will fail with the error DDERR_WASSTILLDRAWING until the next vertical refresh occurs. This behavior is necessary 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 **Flip** on visible surfaces an asynchronous command. A good practice to follow when building applications is to perform all of the non-visual elements of the game after the **Flip** is called and then when the input, audio, gameplay and system memory drawing operations have been completed, begin the drawing tasks that require access to the visible back buffers.

When an application wants to run in a window and still requires a flipping environment, the application will attempt to create a flipping overlay surface. If the hardware does not support overlays, another alternative is to create a primary surface that is page flipping, and whenever the 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 doesn't take any of the application's time since the blits are performed asynchronously. It can, however, consume considerable blitter bandwidth which is dependent on the resolution of the screen and how big the window that is really being page flipped is. As long as the frame rate does not dip below 20 frames a second, GDI will appear to the user to be operating correctly.

Before you instantiate a DirectDraw object, GDI is already using your video 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 backbuffer, GDI will originally be pointing to the video 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 games will start out by creating one large window that covers the entire screen. As long as your game 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 one surface (the original one), and never knows if it is currently the primary surface or a back buffer. If you don't need the GDI screen, then use the above technique. If you do need GDI, you can try this technique:

- Create a Primary surface with 2 back buffers
- Blt the initial Primary Surface (the GDI one) to the middle back buffer
- Flip(NULL) to put GDI into last place and make your initial copy visible

From then on, copy from the GDI buffer to the middle buffer, draw what you want the user to see on that buffer, then pPrimary->Flip(pMiddle), which will keep GDI safely on the bottom and oscillate between the other two buffers.

Overlay Z Order

Overlay Z order is used to determine 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 everything else. Destination colorkeys are only affected by the bits on the primary surface, not by the overlays that may be occluded by other overlays. Source color keys work on the overlay whether or not it has a z order specified. Overlays that do not have a z order specified have unspecified behavior when they overlay the same area on the primary surface. Finally, overlays without a z order specified are assumed to have a z order of zero. The z order of overlays moves from zero, which is just on top of the primary surface, to 4 billion, which is 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. Overlays are not allowed to have the same z order as another overlay.

Palettes and Pixel Formats

DirectDraw enables the creation of multiple palettes, which can be attached to offscreen surfaces. When this is done, the offscreen surfaces no longer share the palette of the primary surface. If an offscreen 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 palettized offscreen surface is created when the primary surface is in 16-bit color mode, then 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 4-bit palettized surface is indexed into a true color palette table, the second is indexed into the primary surface indexed palette table. This second type of palette provides 50% compression and a layer of indirection to the sprites stored using it.

If these surfaces are to be created, the blitter must be able to do the palette replacement during the blit operation. Blting from one palettized surface to another ignores the palette. 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.

Data Structure Summary

Most DirectDraw structures have a "dwSize" field which must be set to the structure's size before the structure is used in a function call.

DDBLTBATCH	This structure is used to pass blit operations to the BltBatch member.
DDBLTFX	This structure is used to pass raster ops, effects, and override information to the **DirectDrawSurface** object member Blt. It is also part of the DDBLTBATCH structure used with the BltBatch member.
DDCAPS	This structure represents the capabilities of the hardware exposed through the DirectDraw object. It contains a DDSCAPS structure which is 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.
DDCOLORKEY	This structure is used to describe a source or destination color key or color space. A color key is specified if the low and high range values are the same.
DDMODEDESC	This structure is returned to the EnumModes' enumeration member to describe the mode that can be created including the monitor frequency and monitor configuration flags.
DDOVERLAYFX	This structure is used to pass override information to the **DIRECTDRAWSURFACE** member UpdateOverlay.
DDPIXELFORMAT	This structure describes the pixel format of a **DIRECTDRAWSURFACE** object.
DDRVAL	Return value used by DirectDraw members.
DDSCAPS	This structure defines the capabilities of DirectDrawSurfaces. It is part of the DDCAPS structure which is used to describe the capabilities of the DirectDraw object.
DDSURFACEDESC	The structure is passed to the **DIRECTDRAW** member CreateSurface to describe the surface that should be created. The relevant fields differ for each type of surface being created.

DDBLTBATCH

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

lprDest

Pointer to a RECT structure that defines the destination for the blit.

lpDDSSrc

Pointer to a DirectDrawSurface that will be the source of the blit.

lprSrc

Pointer to a RECT structure that defines the source rectangle of the blit.

dwFlags

DDBLT_ALPHADEST	Use 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	Use the dwConstAlphaDest field 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	Use the lpDDSAlphaDest field in the DDBLTFX structure as the alpha channel for the destination for this blit.
DDBLT_ALPHAEDGEBLEND	Use the dwAlphaEdgeBlend field in the DDBLTFX structure as the alpha channel for the edges of the image that border the color key colors.
DDBLT_ALPHASRC	Use 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	Use the dwConstAlphaSrc field 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	Use the lpDDSAlphaSrc field in the DDBLTFX structure as the alpha channel for the source for this blit.
DDBLT_ASYNC	Do this blit asynchronously through the FIFO in the order received. If there is no room in the hardware FIFO fail the call.
DDBLT_COLORFILL	Uses the dwFillColor field in the DDBLTFX structure as the RGB color to fill the destination rectangle on the destination surface with.
DDBLT_DDFX	Uses the dwDDFX field in the DDBLTFX structure to specify the effects to use for the blit.
DDBLT_DDROPS	Uses the dwDDROPS field in the DDBLTFX structure to specify the ROPS that are not part of the Win32 API.
DDBLT_KEYDEST	Use the color key associated with the destination surface.
DDBLT_KEYDESTOVERRIDE	Use the dckDestColorkey field in the DDBLTFX structure as the color key for the destination surface.
DDBLT_KEYSRC	Use the color key associated with the source surface.
DDBLT_KEYSRCOVERRIDE	Use the dckSrcColorkey field in the DDBLTFX structure as the color key for the source surface.
DDBLT_ROP	Use the dwROP field in the DDBLTFX structure for the raster operation for this blit. These ROPs are the same as the ones defined in the Win32 API.
DDBLT_ROTATIONANGLE	Use the dwRotationAngle field in the DDBLTFX structure as the angle (specified in 1/100th of a degree) to rotate the surface.
DDBLT_ZBUFFER	Z-buffered blit using the z-buffers attached to the source and destination surfaces and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer opcode.
DDBLT_ZBUFFERDESTCONSTOVERRIDE	Z-buffered blit using the dwConstDest Zfield and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the destination.
DDBLT_ZBUFFERDESTOVERRIDE	Z-buffered blit using the lpDDSDestZBuffer field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the destination.
DDBLT_ZBUFFERSRCCONSTOVERRIDE	Z-buffered blit using the dwConstSrcZ field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the source.

DDBLT_ZBUFFERSRCOVERRIDE	Z-buffered blit using the lpDDSSrcZBuffer field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the source.

lpDDBltFx

Pointer to 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                  wAlphaDestConst;
          LPDIRECTDRAWSURFACE    lpDDSAlphaDest;
   };
   DWORD  dwAlphaSrcConstBitDepth;
   union
   {
          DWORD                  dwAlphaSrcConst;
          LPDIRECTDRAWSURFACE    lpDDSAlphaSrc;
   };
   union
   {
          WORD                                dwFillColor;
          LPDIRECTDRAWSURFACE    lpDDSPattern;
   };
   DDCOLORKEY        dckDestColorkey;
   DDCOLORKEY        dckSrcColorkey;
} DDBLTFX,FAR* LPDDBLTFX;
```

dwSize

Size of the structure. Must be initialized before the structure is used.

dwDDFX FX operations.

DDBLTFX_ARITHSTRETCHY	If stretching, use arithmetic stretching along the Y axis for this blit.
DDBLTFX_MIRRORLEFTRIGHT	Do this blit mirroring the surface left to right. Spin the

	surface around its y-axis.
DDBLTFX_MIRRORUPDOWN	Do this blit mirroring the surface up and down. Spin the surface around its x-axis.
DDBLTFX_NOTEARING	Schedule this blit to avoid tearing.
DDBLTFX_ROTATE180	Do this blit rotating the surface one hundred and eighty degrees clockwise.
DDBLTFX_ROTATE270	Do this blit rotating the surface two hundred and seventy degrees clockwise.
DDBLTFX_ROTATE90	Do this blit rotating the surface ninety degrees clockwise.
DDBLTFX_ZBUFFERRANGE	Do this z blit using dwZBufferLow and dwZBufferHigh as range values specified to limit the bits copied from the source surface.
DDBLTFX_ZBUFFERBASEDEST	Do this z blit adding the dwZBufferBaseDest to each of the sources z values before comparing it with the desting z values.

dwROP
Win32 raster operations.

dwDDROP
DirectDraw raster operations.

dwRotationAngle
Rotation angle for the blit.

dwZBufferOpCode
ZBuffer compares.

dwZBufferLow
Low limit of Z buffer.

dwZBufferHigh
High limit of Z buffer.

dwZBufferBaseDest
Destination base value.

dwZDestConstBitDepth
Bit depth used to specify Z constant for destination.

dwZDestConst
Constant to use as Z buffer for destination.

lpDDSZBufferDest
Surface to use as Z buffer for destination.

dwZSrcConstBitDepth
Bit depth used to specify Z constant for source.

dwZSrcConst
Constant to use as Z buffer for source.

lpDDSZBufferSrc
Surface to use as Z buffer for source.

dwAlphaEdgeBlendBitDepth
Bit depth used to specify constant for alpha edge blend.

dwAlphaEdgeBlend
Alpha for edge blending.

dwReserved
Reserved for future use.

dwAlphaDestConstBitDepth
Bit depth used to specify alpha constant for destination.

dwAlphaDestConst
Constant to use as Alpha Channel.

lpDDSAlphaDest
Surface to use as Alpha Channel.

dwAlphaSrcConstBitDepth
Bit depth used to specify alpha constant for source

dwAlphaSrcConst
Constant to use as Alpha Channel

lpDDSAlphaSrc
Surface to use as Alpha Channel

dwFillColor
Fill color in RGB or Palettized

lpDDSPattern
Surface to use as pattern.

dckDestColorkey
DestColorkey override.

dckSrcColorkey
SrcColorkey 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;
} DDCAPS,FAR* LPDDCAPS;
```

dwSize

Size of structure. Must be initialized before use.

dwCaps **Driver-specific capabilities.**

DDCAPS_3D	Display hardware has 3D acceleration.
DDCAPS_ALIGNBOUNDARYDEST	Indicates that DirectDraw will support only source rectangles whose X axis is aligned on DIRECTDRAWCAPS.dwAlignBoundaryDest boundaries of the surface, respectively.
DDCAPS_ALIGNSIZEDEST	Indicates that DirectDraw will support only source rectangles whose X axis size in BYTEs are DIRECTDRAWCAPS.dwAlignSizeDest multiples, respectively.
DDCAPS_ALIGNBOUNDARYSRC	Indicates that DirectDraw will support only source rectangles whose X axis is aligned on

	DIRECTDRAWCAPS.dwAlignBoundarySrc boundaries of the surface, respectively.
DDCAPS_ALIGNSIZESRC	Indicates that DirectDraw will support only source rectangles whose X axis size in BYTEs are DIRECTDRAWCAPS.dwAlignSizeSrc multiples, respectively.
DDCAPS_ALIGNSTRIDE	Indicates that DirectDraw will create video memory surfaces that have a stride alignment equal to DIRECTDRAWCAPS.dwAlignStrideAlign.
DDCAPS_BANKSWITCHED	Display hardware is bank switched, and potentially very slow at random access to VRAM.
DDCAPS_BLT	Display hardware is capable of blit operations.
DDCAPS_BLTCOLORFILL	Display hardware is capable of color fill with bltter.
DDCAPS_BLTQUEUE	Display hardware is capable of asynchronous blit operations.
DDCAPS_BLTFOURCC	Display hardware is capable of color space conversions during the blit operations.
DDCAPS_BLTSTRETCH	Display hardware is capable of stretching during blit operations.
DDCAPS_GDI	Display hardware is shared with GDI.
DDCAPS_OVERLAY	Display hardware can overlay.
DDCAPS_OVERLAYCANTCLIP	Set if display hardware supports overlays but can not clip them.
DDCAPS_OVERLAYFOURCC	Indicates that overlay hardware is capable of color space conversions during the overlay operation.
DDCAPS_OVERLAYSTRETCH	Indicates that stretching can be done by the overlay hardware.
DDCAPS_PALETTE	Indicates that DirectDraw is capable of creating and supporting **DIRECTDRAWPALETTE** objects for more than the primary surface.
DDCAPS_PALETTECANVSYNC	Indicates that DirectDraw is capable of updating the palette in sync with the veritcal refresh.
DDCAPS_READSCANLINE	Display hardware can return the current scan line.
DDCAPS_STEREOVIEW	Display hardware has stereo vision capabilities. **DDSCAPS_PRIMARYSURFACELEFT** can be created.
DDCAPS_VBI	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 **OverlayZOrder** as a z value for overlays to control their layering.
DDCAPS_COLORKEY	Obsolete flag that represents colorkey capabilities in either overlay hardware or blit hardware. Will be replaced with **DDCAPS_OVERLAYCOLORKEY** and **DDCAPS_BLTCOLORKEY**.
DDCAPS_ALPHA	Display hardware supports alpha channel during blit operations.
DDCAPS_COLORKEY_HWASSIST	Colorkey is hardware assisted.
DDCAPS_NOHARDWARE	No hardware support at all.

dwCaps2 **More driver-specific capabilities.**

DDCAPS2_CERTIFIED	Display hardware is certified.

dwCKeyCaps **Color key capabilities.**

DDCKEYCAPS_DESTBLT	Supports transparent blitting using a color key to identify the replaceable bits of the destination surface for RGB colors.
DDCKEYCAPS_DESTBLTCLRSPACE	Supports transparent blitting using a color space to identify the replaceable bits of the destination surface for RGB colors.
DDCKEYCAPS_DESTBLTCLRSPACEYUV	Supports transparent blitting using a color space to identify the replaceable bits of the destination surface for YUV colors.
DDCKEYCAPS_DESTBLTYUV	Supports transparent blitting using a color key to identify the replaceable bits of the destination surface for YUV colors.
DDCKEYCAPS_DESTOVERLAY	Supports overlaying using colorkeying of the replaceable bits of the surface being overlayed for

	RGB colors.
DDCKEYCAPS_DESTOVERLAYCLRSPACE	Supports a color space as the color key for the destination for RGB colors.
DDCKEYCAPS_DESTOVERLAYCLRSPACEYUV	Supports a color space as the color key for the destination for YUV colors.
DDCKEYCAPS_DESTOVERLAYONEACTIVE	Supports only one active destination color key value for visible overlay surfaces.
DDCKEYCAPS_DESTOVERLAYYUV	Supports overlaying using colorkeying of the replaceable bits of the surface being overlayed for YUV colors.
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 overlays using the color key for the source with this overlay surface for RGB colors.
DDCKEYCAPS_SRCOVERLAYCLRSPACE	Supports overlays using a color space as the source color key for the overlay surface for RGB colors.
DDCKEYCAPS_SRCOVERLAYCLRSPACEYUV	Supports overlays 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 overlays using the color key for the source with this overlay surface for YUV colors.

dwFXCaps **Driver-specific stretching and effects capabilities.**

DDFXCAPS_BLTARITHSTRETCHY	Uses arithmetic operations to stretch and shrink surfaces during blit rather than pixel doubling techniques. Along the Y axis.
DDFXCAPS_BLTARITHSTRETCHYN	Uses arithmetic operations to stretch and shrink surfaces during blit rather than pixel doubling techniques. Along the Y axis. Only works for x1, x2, etc.
DDFXCAPS_BLTMIRRORLEFTRIGHT	Supports mirroring left to right in blit.
DDFXCAPS_BLTMIRRORUPDOWN	Supports mirroring top to bottom in blit.
DDFXCAPS_BLTROTATION	Supports arbitrary rotation.
DDFXCAPS_BLTROTATION90	Supports 90 degree rotations.
DDFXCAPS_BLTSHRINKX	Supports arbitrary shrinking of a surface along the x axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSHRINKXN	Supports integer shrinking (1x,2x,) of a surface along the x axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSHRINKY	Supports arbitrary shrinking of a surface along the y axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSHRINKYN	Supports integer shrinking (1x,2x,) of a surface along the y axis (vertical direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSTRETCHX	Supports arbitrary stretching of a surface along the x axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSTRETCHXN	Supports integer stretching (1x,2x,) of a surface along the x axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSTRETCHY	Supports arbitrary stretching of a surface along the y axis (horizontal direction). This flag is only valid for blit operations.
DDFXCAPS_BLTSTRETCHYN	Supports integer stretching (1x,2x,) of a surface along the y axis (vertical direction). This flag is only valid for blit operations.
DDFXCAPS_OVERLAYARITHSTRETCHY	Uses arithmetic operations to stretch and shrink surfaces during overlay rather than pixel doubling techniques. Along the Y axis.

DDFXCAPS_OVERLAYARITHSTRETCHYN	Uses arithmetic operations to stretch and shrink surfaces during overlay rather than pixel doubling techniques. Along the Y axis. Only works for x1, x2, etc.
DDFXCAPS_OVERLAYSHRINKX	Supports arbitrary shrinking of a surface along the x axis (horizontal direction). 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 (1x,2x,) of a surface along the x axis (horizontal direction). 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 (vertical direction). 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 (1x,2x,) of a surface along the y axis (vertical direction). 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 (horizontal direction). 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 (1x,2x,) of a surface along the x axis (horizontal direction). 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 (vertical direction). 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 (1x,2x,) of a surface along the y axis (vertical direction). 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_OVERLAYMIRRORLEFTRIGHT	Supports mirroring of overlays across the vertical axis.
DDFXCAPS_OVERLAYMIRRORUPDOWN	Supports mirroring of overlays across the horizontal axis.

dwFXAlphaCaps Driver specific alpha capabilities.

DDFXALPHACAPS_BLTALPHAEDGEBLEND	Supports alpha blending around the edge of a source color keyed surface. For Blt.
DDFXALPHACAPS_BLTALPHAPIXELS	Supports alpha information in the 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. (0 is transparent.) For Blt.
DDFXALPHACAPS_BLTALPHAPIXELSNEG	Supports alpha information in the 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. (0 is opaque.) This flag can only be set if DDCAPS_ALPHA is set. For Blt.
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. (0 is transparent.) For Blt.
DDFXALPHACAPS_BLTALPHASURFACESNEG	The depth of the alpha channel data can range can be 1,2,4, or 8. The NEG suffix indicates that this alpha channel becomes more transparent as

	the alpha value increases. (0 is opaque.) This flag can only be set if DDFXCAPS_ALPHASURFACES is set. For Blt.
DDFXALPHACAPS_OVERLAYALPHAEDGEBLEND	Supports alpha blending around the edge of a source color keyed surface. For Overlays.
DDFXALPHACAPS_OVERLAYALPHAPIXELS	Supports alpha information in the 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. (0 is transparent.) For Overlays.
DDFXALPHACAPS_OVERLAYALPHAPIXELSNEG	Supports alpha information in the 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. (0 is opaque.) This flag can only be set if DDFXCAPS_ALPHAPIXELS is set. 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. (0 is transparent.) For Overlays.
DDFXALPHACAPS_OVERLAYALPHASURFACESNEG	The depth of the alpha channel data can range can be 1,2,4, or 8. The NEG suffix indicates that this alpha channel becomes more transparent as the alpha value increases. (0 is opaque.) This flag can only be set if DDFXCAPS_ALPHASURFACES is set. For Overlays.

dwPalCaps **Palette capabilities.**

DDPCAPS_4BIT	Index is 4 bits. There are sixteen color entries in the palette table.
DDPCAPS_8BITENTRIES	Index is onto an 8 bit color index. This field is only valid with the DDPCAPS_4BIT capability and the target surface is in 8bpp. Each color entry is one byte long and is an index into destination surface's 8bpp palette.
DDPCAPS_8BIT	Index is 8 bits. There are 256 color entries in the palette table.
DDPCAPS_ALLOW256	This palette can have all 256 entries defined.
DDPCAPS_INITIALIZE	Indicates that this **DIRECTDRAWPALETTE** should use the palette color array passed into the **lpDDColorArray** parameter to initialize the **DIRECTDRAWPALETTE** object.
DDPCAPS_PRIMARYSURFACE	This palette is the one attached to the primary surface. Changing this table has immediate effect on the display unless DDPAL_VSYNC is specified and supported.
DDPCAPS_PRIMARYSURFACELEFT	This palette is the one attached to the primary surface left. Changing this table has immediate effect on the display unless DDPAL_VSYNC is specified and supported.
DDPCAPS_VSYNC	This palette can have modifications to it synced with the monitors refresh rate.

dwSVCaps **Stereo vision capabilities.**

DDSVCAPS_ENIGMA	The stereo view is accomplished via Enigma encoding.
DDSVCAPS_FLICKER	The stereo view is accomplished via high frequency flickering.
DDSVCAPS_REDBLUE	The stereo view is accomplished via red and blue filters applied to the left and right eyes. All images must adapt their color spaces for this process.
DDSVCAPS_SPLIT	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 video memory.

dwVidMemFree
Amount of free video 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]
ROPS supported.

ddsCaps
DDSCAPS structure with general capabilities.

dwMinOverlayStretch
Minimum overlay stretch factor multiplied by 1000

dwMaxOverlayStretch
Maximum overlay stretch factor multiplied by 1000

dwMinLiveVideoStretch
Minimum live video stretch factor multiplied by 1000, eg 1000 == 1.0, 1300 == 1.3

dwMaxLiveVideoStretch
Maximum live video stretch factor multiplied by 1000, eg 1000 == 1.0, 1300 == 1.3

dwMinHwCodecStretch
Minimum hardware codec stretch factor multiplied by 1000, eg 1000 == 1.0, 1300 == 1.3

dwMaxHwCodecStretch;
Maximum hardware codec stretch factor multiplied by 1000, eg 1000 == 1.0, 1300 == 1.3

dwReserved1,dwReserved2,dwReserved3
Reserved.

dw...BitDepths

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;
```

dwColorSpaceLowValue
The low value of the color range that is to be used as the color key, inclusive.

dwColorSpaceHighValue
The high value of the color range that is to be used as the color key, inclusive.

DDMODEDESC

```
typedef struct {
   DWORD             dwSize;
   DWORD             dwFlags;
   DWORD             dwMonitorFrequency;
   DDSURFACEDESC     dsdSurfaceDesc;

} DDMODEDESC, FAR* LPDDMODEDESC;
```

dwSize
Size of the structure. Must be initialized before use.

dwFlags
Enumeration flags.

dwMonitorFrequency
Frequency of the monitor in this mode.

dsdSurfaceDesc
Surface description.

DDOVERLAYFX

```
typedef struct _DDOVERLAYFX{
   DWORD  dwSize;
   DWORD  dwAlphaEdgeBlendBitDepth;
   DWORD  dwAlphaEdgeBlend;
   DWORD  dwReserved;
   DWORD  dwAlphaDestConstBitDepth;
   union
   {
          DWORD                   dwAlphaDestConst;
          LPDIRECTDRAWSURFACE     lpDDSAlphaDest;
   };
   DWORD  dwAlphaSrcConstBitDepth;
   {
          DWORD                   dwAlphaSrcConst;
          LPDIRECTDRAWSURFACE     lpDDSAlphaSrc;
   };
   DDCOLORKEY        dckDestColorkey;
   DDCOLORKEY        dckSrcColorkey;
   DWORD  dwDDFX;
   DWORD  dwFlags;
} DDOVERLAYFX,FAR *LPDDOVERLAYFX;
```

dwSize
Size of structure. Must be initialized before use.

dwAlphaEdgeBlendBitDepth
Bit depth used to specify constant for alpha edge blend.

dwAlphaEdgeBlend
Constant to use as alpha for edge blend.

dwReserved
Reserved.

dwAlphaDestConstBitDepth
Bit depth used to specify alpha constant for destination

dwAlphaDestConst
Constant to use as alpha channel for destination.

lpDDSAlphaDest
Pointer to a surface to use as alpha channel for destination.

dwAlphaSrcConstBitDepth
Bit depth used to specify alpha constant for source.

dwAlphaSrcConst
Constant to use as alpha channel for source.

lpDDSAlphaSrc
Pointer to a surface to use as alpha channel for source.

dckDestColorkey
DestColorkey override.

dckSrcColorkey
DestColorkey override.

dwDDFX **Overlay FX Flags**

DDOVERFX_ARITHSTRETCHY	If stretching, use arithmetic stretching along the Y axis

for this overlay.

DDOVERFX_MIRRORLEFTRIGHT	Mirror the overlay across the vertical axis.
DDOVERFX_MIRRORUPDOWN	Mirror the overlay across the horizontal axis.

dwFlags

Flags

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;
```

dwSize

Size of the structure. Must be initialized prior to use.

dwFlags

DDPF_ALPHAPIXELS	The surface has alpha channel information in the pixel format.
DDPF_ALPHA	The pixel format describes an alpha only surface.
DDPF_FOURCC	The FourCC code is valid.
DDPF_PALETTEINDEXED4	The surface is 4-bit color indexed.
DDPF_PALETTEINDEXED4TO8	The surface is 4-bit color indexed to an 8-bit palette.
DDPF_PALETTEINDEXED8	The surface is 8-bit color indexed.
DDPF_RGB	The RGB data in the pixel format structure is valid.
DDPF_COMPRESSED	The surface will accept pixel data in the specified format and compress it during the write.
DDPF_RGBTOYUV	The surface will accept RGB data and translate it during the write 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.

DDRVAL

```
typedef long       DDRVAL;
```

DDRVAL

DirectDraw return value

DDSCAPS

```
typedef struct _DDSCAPS{
   DWORD  dwCaps;
} DDSCAPS,  FAR* LPDDSCAPS;
```

dwCaps

DDSCAPS_3D	Indicates that this surface is a front buffer, back buffer, or texture map that is being used in conjunction with a 3DDDI or Direct3D HAL.
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 (e.g. RGBA or YUVA).
DDSCAPS_BACKBUFFER	Indicates that this surface is a backbuffer. It is generally set by CreateSurface when the DDSCAPS_FLIP capability bit is set. It indicates that this surface is THE back buffer of a surface flipping structure. DirectDraw supports N surfaces in a surface flipping structure. Only the surface that immediately precedes the DDSCAPS_FRONTBUFFER has this capability bit set. The other surfaces are identified as back buffers by the presence of the DDSCAPS_FLIP capability, their attachment order, and the absence of the DDSCAPS_FRONTBUFFER and DDSCAPS_BACKBUFFER capabilities. The bit is sent to CreateSurface when a standalone back buffer is being created. This surface could be attached to a front buffer and/or back buffers to form a flipping surface structure after the CreateSurface call. See AddAttachedSurface for a detailed description of the behaviors in this case.
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 it is passed to CreateSurface the DDSCAPS_FRONTBUFFER and DDSCAP_BACKBUFFER bits are not set. They are set by CreateSurface on the resulting creations. The dwBackBufferCount field in the DDSURFACEDESC structure must be set to at least 1 in order for the CreateSurface call to succeed. The DDSCAPS_COMPLEX capability must always be set when creating multiple surfaces through CreateSurface.
DDSCAPS_FRONTBUFFER	Indicates that this surface is THE front buffer of a surface flipping structure. It is generally set by CreateSurface when the DDSCAPS_FLIP capability bit is set. If this capability is sent to CreateSurface then 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. See AddAttachedSurface for a detailed description of the behaviors in this case.
DDSCAPS_HWCODEC	Indicates surface should be able to have a stream decompressed to it by the hardware.
DDSCAPS_LIVEVIDEO	Indicates surface should be able to receive live video.
DDSCAPS_MODEX	Surface is a 320x200 or 320x240 ModeX surface.
DDSCAPS_OFFSCREENPLAIN	Indicates that this surface is any offscreen surface that is not an overlay, texture, zbuffer, front buffer, back buffer, or alpha surface. It is used to identify plain vanilla surfaces.
DDSCAPS_OWNDC	Indicates surface will have a DC associated long term.
DDSCAPS_OVERLAY	Indicates that this surface is an overlay. It may or may not be directly visible depending on whether or not it is currently being overlayed onto the primary surface. DDSCAPS_VISIBLE can be used to determine whether or not it is being overlayed at the

	moment.
DDSCAPS_PALETTE	Indicates that unique DirectDrawPalette objects can be created and attached to this surface.
DDSCAPS_PRIMARYSURFACE	Indicates that this surface is the primary surface. The primary surface represents what the user is seeing at the moment.
DDSCAPS_PRIMARYSURFACELEFT	Indicates that this surface is the primary surface for the left eye. The primary surface for the left eye represents what the user is seeing at the moment with the user's left eye. When this surface is created the DDSCAPS_PRIMARYSURFACE represents what the user is seeing with the user's right eye.
DDSCAPS_SYSTEMMEMORY	Indicates that this surface memory was allocated in system memory.
DDSCAPS_TEXTUREMAP	Indicates that this surface can be used as a 3D texture. It does not indicate whether or not the surface is being used for that purpose.
DDSCAPS_VIDEOMEMORY	Indicates that this surface exists in video 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 overlayed and texture maps while they are being textured.
DDSCAPS_WRITEONLY READ ONLY.	Indicates that only writes are permitted to the surface. Read accesses from the surface may or may not generate a protection fault, but the results of a read from this surface will not be meaningful.
DDSCAPS_ZBUFFER	Indicates that this surface is the z buffer. The z buffer does not contain displayable information. Instead, it contains bit depth information that is used to determine which pixels are visible and which are obscured.

DDSURFACEDESC

Structure

```
typedef struct _DDSURFACEDESC{
   DWORD             dwSize;
   DWORD             dwFlags;
   DWORD             dwHeight;
   DWORD             dwWidth;
   LONG               lPitch;
   DWORD             dwBackBufferCount;
   DWORD             dwZBufferBitDepth;
   DWORD             dwAlphaBitDepth;
   LPVOID            lpSurface;
   DDCOLORKEY        ddckCKDestOverlay;
   DDCOLORKEY        ddckCKDestBlt;
   DDCOLORKEY        ddckCKSrcOverlay;
   DDCOLORKEY        ddckCKSrcBlt;
   DDPIXELFORMAT     ddpfPixelFormat;
   DDSCAPS            ddsCaps;
} DDSURFACEDESC,  FAR* LPDDSURFACEDESC;
```

dwSize

Size of the structure. Must be initialized prior to use.

dwFlags

DDSD_DDSCAPS	ddsCaps field is valid.
DDSD_HEIGHT	dwHeight field is valid.
DDSD_WIDTH	dwWidth field is valid.
DDSD_PITCH	lPitch is valid.
DDSD_BACKBUFFERCOUNT	dwBackBufferCount is valid.

DDSD_ZBUFFERBITDEPTH	dwZBufferBitDepth is valid.
DDSD_ALPHABITDEPTH	dwAlphaBitDepth is valid.
DDSD_LPSURFACE	lpSurface is valid.
DDSD_PIXELFORMAT	ddpfPixelFormat is valid.
DDSD_CKDESTOVERLAY	ddckCKDestOverlay is valid.
DDSD_CKDESTBLT	ddckCKDestBlt is valid.
DDSD_CKSRCOVERLAY	ddckCKSrcOverlay is valid.
DDSD_CKSRCBLT	ddckCKSrcBlt is valid.
DDSD_ALL	All input fields are 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.

dwZBufferBitDepth
Depth of Z buffer.

dwAlphaBitDepth
Depth of alpha buffer.

lpSurface
Pointer to 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.

DD OK

The OK message indicates success and is returned when any DirectDraw related member has performed the action requested of it.

DD_OK

Status OK, request completed successfully.

DirectDraw Enumeration Callback Return Codes

EnumCallback returns are used to control the flow of DirectDraw and DirectDrawSurface enumerations. They can only be returned by the enumeration members.

DDENUMRET_CANCEL
Stop the enumeration.

DDENUMRET_OK
Continue the enumeration.

DirectDraw Error Return Codes

Errors are represented by negative values and cannot be combined. This table lists the failures that can be returned by all DirectDraw, DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper members. See the individual member descriptions for a list of the error codes each one is capable of returning.

DDERR_ALREADYINITIALIZED
This object is already initialized.

DDERR_BLTFASTCANTCLIP
Return if a clipper object is attached to the source surface passed into a BltFast call.

DDERR_CANNOTATTACHSURFACE
This surface can not be attached to the requested surface.

DDERR_CANNOTDETACHSURFACE
This surface can not be detached from the requested surface.

DDERR_CANTCREATEDC
Windows can not create any more DCs

DDERR_CANTDUPLICATE
Can't duplicate primary & 3D surfaces, or surfaces that are implicitly created.

DDERR_CLIPPERISUSINGHWND
An attempt was made to set a cliplist for a clipper object that is already monitoring an hwnd.

DDERR_COLORKEYNOTSET
No src color key specified for this operation.

DDERR_CURRENTLYNOTAVAIL
Support is currently not available.

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
Generic failure.

DDERR_HEIGHTALIGN
Height of rectangle provided is not a multiple of reqd alignment.

DDERR_HWNDALREADYSET
The CooperativeLevel HWND has already been set. It can not be reset while the process has surfaces or palettes created.

DDERR_HWNDSUBCLASSED
HWND used by DirectDraw CooperativeLevel has been subclassed, this prevents DirectDraw from restoring state.

DDERR_IMPLICITLYCREATED
This surface can not be restored because it is an implicitly created surface.

DDERR_INCOMPATIBLEPRIMARY
Unable to match primary surface creation request with existing primary surface.

DDERR_INVALIDCAPS
One or more of the caps bits passed to the callback are incorrect.

DDERR_INVALIDCLIPLIST
DirectDraw does not support the provided cliplist.

DDERR_INVALIDDIRECTDRAWGUID
The GUID passed to DirectDrawCreate 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 function are incorrect.

DDERR_INVALIDPIXELFORMAT
The pixel format was invalid as specified.

DDERR_INVALIDPOSITION
Returned when the position of the overlay on the destination is no longer legal for that destination.

DDERR_INVALIDRECT
Rectangle provided was invalid.

DDERR_LOCKEDSURFACES
Operation could not be carried out because one or more surfaces are locked.

DDERR_NO3D
There is no 3D present.

DDERR_NOALPHAHW
Operation could not be carried out because there is no alpha accleration hardware present or available.

DDERR_NOANTITEARHW
Operation could not be carried out because there is no hardware support for synchronizing blits to avoid tearing.

DDERR_NOBLTHW
No blitter hardware present.

DDERR_NOBLTQUEUEHW
Operation could not be carried out because there is no hardware support for asynchronous blitting.

DDERR_NOCLIPLIST
No cliplist available.

DDERR_NOCLIPPERATTACHED
No clipper object attached to surface object.

DDERR_NOCOLORCONVHW
Operation could not be carried out because there is no color conversion hardware present or available.

DDERR_NOCOLORKEY
Surface doesn't currently have a color key

DDERR_NOCOLORKEYHW
Operation could not be carried out because there is no hardware support of the destination color key.

DDERR_NOCOOPERATIVELEVELSET
Create function called without DirectDraw object method SetCooperativeLevel being called.

DDERR_NODC
No DC was ever created for this surface.

DDERR_NODDROPSHW
No DirectDraw ROP hardware.

DDERR_NODIRECTDRAWHW
A hardware-only DirectDraw object creation was attempted but the driver did not support any hardware.

DDERR_NOEMULATION
Software emulation not available.

DDERR_NOEXCLUSIVEMODE
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
There is no GDI present.

DDERR_NOHWND
Clipper notification requires an HWND or no HWND has previously been set as the CooperativeLevel HWND.

DDERR_NOMIRRORHW
Operation could not be carried out because there is no hardware present or available.

DDERR_NOOVERLAYDEST
Returned when GetOverlayPosition is called on an overlay that UpdateOverlay has never been called on to establish a destination.

DDERR_NOOVERLAYHW
Operation could not be carried out because there is no overlay hardware present or available.

DDERR_NOPALETTEATTACHED
No palette object attached to this surface.

DDERR_NOPALETTEHW
No hardware support for 16 or 256 color palettes.

DDERR_NORASTEROPHW
Operation could not be carried out because there is no appropriate raster op hardware present or available.

DDERR_NOROTATIONHW
Operation could not be carried out because there is no rotation hardware present or available.

DDERR_NOSTRETCHHW
Operation could not be carried out because there is no hardware support for stretching.

DDERR_NOT4BITCOLOR
DirectDrawSurface is not in 4 bit color palette and the requested operation requires 4 bit color palette.

DDERR_NOT4BITCOLORINDEX
DirectDrawSurface is not in 4 bit color index palette and the requested operation requires 4 bit color index palette.

DDERR_NOT8BITCOLOR
DirectDrawSurface is not in 8 bit color mode and the requested operation requires 8 bit color.

DDERR_NOTAOVERLAYSURFACE
Returned when an overlay member is called for a non-overlay surface.

DDERR_NOTEXTUREHW
Operation could not be carried out because there is no texture mapping hardware present or available.

DDERR_NOTFLIPPABLE
An attempt has been made to flip a surface that is not flippable.

DDERR_NOTFOUND
Requested item was not found.

DDERR_NOTLOCKED
Surface was not locked. An attempt to unlock a surface that was not locked at all, or by this process, has been attempted.

DDERR_NOTPALETTIZED
The surface being used is not a palette-based surface.

DDERR_NOVSYNCHW
Operation could not be carried out because there is no hardware support for vertical blank synchronized operations.

DDERR_NOZBUFFERHW
Operation could not be carried out because there is no hardware support for zbuffer blitting.

DDERR_NOZOVERLAYHW
Overlay surfaces could not 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 memory to perform the operation.

DDERR_OVERLAYCANTCLIP
The hardware does not support clipped overlays.

DDERR_OVERLAYCOLORKEYONLYONEACTIVE
Can only have ony color key active at one time for overlays.

DDERR_OVERLAYNOTVISIBLE
Returned when GetOverlayPosition is called on a hidden overlay.

DDERR_PALETTEBUSY
Access to this palette is being refused because the palette is already locked by another thread.

DDERR_PRIMARYSURFACEALREADYEXISTS
This process already has created a primary surface.

DDERR_REGIONTOOSMALL
Region passed to Clipper::GetClipList is too small.

DDERR_SURFACEALREADYATTACHED
This surface is already attached to the surface it is being attached to.

DDERR_SURFACEALREADYDEPENDENT
This surface is already a dependency of the surface it is being made a dependency of.

DDERR_SURFACEBUSY
Access to this surface is being refused because the surface is already locked by another thread.

DDERR_SURFACEISOBSCURED
Access to surface refused because the surface is obscured.

DDERR_SURFACELOST
Access to this surface is being refused because the surface memory is gone. The DirectDrawSurface object representing this surface should have **Restore** called on it.

DDERR_SURFACENOTATTACHED
The requested surface is not attached.

DDERR_TOOBIGHEIGHT
Height requested by DirectDraw is too large.

DDERR_TOOBIGSIZE
Size requested by DirectDraw is too large -- the individual height and width are OK.

DDERR_TOOBIGWIDTH
Width requested by DirectDraw is too large.

DDERR_UNSUPPORTED
Action not supported.

DDERR_UNSUPPORTEDFORMAT
FOURCC format requested is unsupported by DirectDraw.

DDERR_UNSUPPORTEDMASK
Bitmask in the pixel format requested is unsupported by DirectDraw.

DDERR_VERTICALBLANKINPROGRESS
Vertical blank is in progress.

DDERR_WASSTILLDRAWING
Informs DirectDraw that the previous Blt which is transfering information to or from this Surface is incomplete.

DDERR_WRONGMODE

This surface can not be restored because it was created in a different mode.

DDERR_XALIGN

Rectangle provided was not horizontally aligned on required boundary.

APIs

The DirectDraw APIs are used to initiate control of the video memory through the DirectDraw interface. There are only two. The first one, **DirectDrawCreate**, is used to instantiate a DirectDraw object that represents a specific piece of display hardware. The second one, **DirectDrawEnumerate**, is used to obtain a list of all the DirectDraw objects installed on the system. This is the mechanism DirectDraw uses to support multiple pieces of display hardware. To support multiple display devices the application need only select a specific DirectDraw object and instantiate it.

See:

DirectDrawCreate

DirectDrawEnumerate

DirectDrawCreate

Create an instance of a DirectDraw object. It attempts to initialize a **DirectDraw** object and sets a pointer to it if it was successful. Calling the **DirectDraw** member GetCaps immediately after initialization is advised to determine to what extent this object is hardware accelerated.

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

Parameters

lpGUID
Points to the **GUID** representing the driver that should be created. **NULL** is always the active display driver.

lplpDD
Points to a pointer to be initialized with a valid DirectDraw pointer if the call succeeds.

pUnkOuter
This parameter is provided for future compability with COM aggregation features. Presently, however, **DirectDrawCreate** will return an error if it is anything but NULL.

Return Values

DD_OK

DDERR_INVALIDPARAMS	**DDERR_INVALIDDIRECTDRAWGUID**
DDERR_GENERIC	**DDERR_OUTOFMEMORY**
DDERR_NODIRECTDRAWHW	**DDERR_DIRECTDRAWALREADYCREATED**

DirectDrawEnumerate

Enumerate the DirectDraw objects installed on the system. The **NULL GUID** entry is always used to identify the primary display device that is shared with GDI.

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

Parameters

lpCallback

Points to a callback function that will be called with a description of each DirectDraw enabled HAL (Hardware Abstraction Layer) installed in the system.

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

lpGUID

Pointer to the unique identifier of the DirectDraw object.

lpDriverDescription

Pointer to a string containing the driver description.

lpDriverName

Pointer to a string containing the driver name.

lpContext

Pointer to a caller-defined context.

Return Value

DDENUMRET_OK	**Continue the enumeration**
DDENUMRET_CANCEL	**Stop the enumeration**

lpContext

Points to a caller-defined context that will be passed to the enumeration callback each time it is called.

Return Values

DD_OK	**DDERR_INVALIDPARAMS**

Overview

The DirectDraw object represents the display hardware. The object is hardware-accelerated if the display device for which it was instantiated has hardware acceleration. The three objects which can be created by a DirectDraw object are the DirectDrawSurface, DirectDrawPalette, and DirectDrawClipper objects. For a detailed discussion of these objects see Chapter 1: Overview, Chapter 6: DirectDrawSurface Member Reference, Chapter 7: DirectDrawPalette Member Reference, and Chapter 8: DirectDrawClipper Member Reference.

It is possible to have more than one DirectDraw object instantiated at one time. The simplest example of this would be two monitors. Windows'95 does not support dual monitors natively. However, it is possible to write a DirectDraw HAL for each of the display devices. The display device that Windows'95 and GDI are aware of is the default one which will be used when the default DirectDraw object is instantiated. The display device that Windows'95 and GDI do not know about will be addressed by another, independent DirectDraw object that must be created using its identifying GUID. This GUID can be obtained through the DirectDraw API DirectDrawEnumerate.

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

Changing the display mode is an important piece of DirectDraw's 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 their surface memory and become inoperative when the mode is changed. These surfaces' memory must be reallocated using the Restore member.

DirectDraw Member Implementation

The members **AddRef**, **QueryInterface**, and **Release** are from the standard **COM** interface **IUnknown** which all **COM** interfaces inherit. These three members allow additional interfaces to be added to the DirectDraw object without affecting the functionality of the original interface.

DirectDraw Members

AddRef
Compact
CreateClipper
CreatePalette
CreateSurface
DuplicateSurface
EnumDisplayModes
EnumSurfaces
FlipToGDISurface
GetCaps
GetDisplayMode
GetFourCCCodes
GetGDISurface
GetMonitorFrequency
GetScanLine
GetVerticalBlankStatus
Initialize
QueryInterface
Release
RestoreDisplayMode
SetCooperativeLevel
SetDisplayMode
WaitForVerticalBlank

AddRef

Increase the reference count of the **DirectDraw** object. This member is part of the **IUnknown** interface inherited by **DirectDraw**. When the **DirectDraw** object is initially created, its reference count is set to one. Each time a new application binds to the **DirectDraw** object, or a previously bound application binds to a different **COM** interface of the **DirectDraw** object, the reference count is increased by one. The **DirectDraw** object deallocates itself when its reference count goes to zero. The Release member is used to notify the **DirectDraw** object that an application is no longer bound to the **DirectDraw** object.

```
DWORD AddRef(
    LPDIRECTDRAW lpDD)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

Initialize, **QueryInterface**, **Release**

Compact

At present this member is only a stub -- it has not yet been implemented.

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

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

```
HRESULT Compact(
    LPDIRECTDRAW lpDD)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

Return Values

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

CreateClipper

Create a DirectDrawClipper object. The DirectDrawClipper can be attached to a DirectDrawSurface if desired, and used during Blt, BltBatch, and UpdateOverlay operations.

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

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwFlags
Not currently used. Zero is the only valid value.

lplpDDClipper
Points to a pointer which will be filled in with the address of the new **DirectDrawClipper** object if the **CreateClipper** member is successful.

pUnkOuter
This parameter is provided for future compability with COM aggregation features. Presently, however, CreateClipper will return an error if it is anything but NULL.

Return Values

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

See Also

GetClipper, **SetClipper**

CreatePalette

Create a **DIRECTDRAWPALETTE** object for this **DirectDraw** object.

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

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwFlags

DDPCAPS_4BIT
Index is 4 bits. There are sixteen color entries in the palette table.

DDPCAPS_8BITENTRIES
Index is onto an 8 bit color index. This field is only valid with the DDPCAPS_4BITINDEX capability and the target surface is in 8bpp. Each color entry is one byte long and is an index into destination surface's 8bpp palette.

DDPCAPS_8BIT
Index is 8 bits. There are 256 color entries in the palette table.

DDPCAPS_ALLOW256
This palette can have all 256 entries defined.

DDPCAPS_INITIALIZE
Indicates that this **DIRECTDRAWPALETTE** should use the palette color array passed into the **lpDDColorArray** parameter to initialize the **DIRECTDRAWPALETTE** object.

lpColorTable
Points to an array of 16 or 256 **PALETTEENTRY** structures that should be used to initialize this **DIRECTDRAWPALETTE** object.

lplpDDPalette
Points to a pointer which will be filled in with the address of the new **DIRECTDRAWPALETTE** object if the **CreatePalette** member is successful.

pUnkOuter
This parameter is provided for future compability with COM aggregation features. Presently, however, CreatePalette will return an error if it is anything but NULL.

Return Values

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

CreateSurface

Create a DirectDrawSurface object for this DirectDraw object. The DirectDrawSurface object represents a ***Surface*** (pixel memory), which usually resides in video card memory but may exist in system memory if video memory is exhausted or if explicitly requested. The member will fail if the hardware cannot provide support for the capabilities requested or has previously allocated those resources to another DirectDrawSurface object.

CreateSurface usually creates one DirectDrawSurface object. If the **DDSCAPS_FLIP** flag, in the **dwCaps** field of the DDCAPS structure, which is included in the DDSURFACEDESC structure, is set, however, **CreateSurface** will create several DirectDrawSurface objects which are referred to collectively as a *Complex Structure*. The additional surfaces created are also referred to as "implicit" surfaces.

Note DirectDraw does not permit the creation of surfaces that are wider than the primary surface.

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

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpSurfaceDesc
Points to the DDSURFACEDESC structure which describes the requested **Surface**.

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

pUnkOuter
This parameter is provided for future compability with COM aggregation features. Presently, however, CreateSurface will return an error if it is anything but NULL.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_OUTOFVIDEOMEMORY**
DDERR_NODIRECTDRAWHW	**DDERR_NOCOOPERATIVELEVELSET**
DDERR_INVALIDCAPS	**DDERR_INVALIDPIXELFORMAT**
DDERR_NOALPHAHW	**DDERR_NOFLIPHW**
DDERR_NOZBUFFERHW	**DDERR_NOEXCLUSIVEMODE**
DDERR_OUTOFMEMORY	**DDERR_PRIMARYSURFACEALREADYEXISTS**
DDERR_NOEMULATION	**DDERR_INCOMPATIBLEPRIMARY**

Comments

Here are examples of legal 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 which is being used by GDI. Consequently, while all other types of surfaces require dwHeight and dwWidth, a primary surface *must not* have them specified, even if you know they are the same dimensions as the existing surface.

The fields of the DDSURFACEDESC structure, ddsd below, relevent to the creation of the **Primary Surface** are filled in.

```
DDSURFACEDESC  ddsd;
ddsd.dwSize = sizeof( ddsd );
//Tell DDRAW which fields are valid
```

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

Scenario 2:

Create a simple offscreen surface, of the type that might be used to cache bitmaps which will later be composed with the bltter. A height and width are required for all surfaces except primary surfaces. The fields in the DDSURFACEDESC structure, ddsd below, relevent to the creation of a simple **offscreen surface** are filled in.

```
DDSURFACEDESC  ddsd;
ddsd.dwSize = sizeof( ddsd );
//Tell DDRAW which fields are valid
ddsd.dwFlags = DDSD_DDSCAPS | DDSD_HEIGHT | DDSD_WIDTH;
//Ask for a simple offscreen surface, sized 100 by 100 pixels
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;
dwHeight = 100;
dwWidth = 100;
```

DirectDraw will create this surface in video memory unless it will not fit, in which case the surface will be 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. In this case, an error will be returned if the surface cannot be created in the specified location.

DirectDraw also allows for the creation of **Complex Surfaces**. A complex surface is actually a set of surfaces created with a single call to the **CreateSurface** member. If the DDSCAPS_COMPLEX flag is set in the **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 **Release** will release all surfaces in the structure, and a single call to **Restore** will restore them all.

Scenario 3:

Perhaps the most useful complex surface is one composed of a Primary Surface and one or more back buffers that form a surface flipping environment. The fields in the DDSURFACEDESC structure, ddsd below, relevent 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_DDSCAPS | 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 Flip member will exchange the surface memory of the primary surface and the back buffer. If a BackBufferCount of "2" had been specified, two back buffers would have been created, and each call to **Flip** would have rotated the surfaces in a circular pattern, providing a triple buffered flipping environment.

DuplicateSurface

Duplicate a DirectDrawSurface. This member creates a new DirectDrawSurface object which points to the same surface memory as an existing DirectDrawSurface object. This duplicate can be used just like the original. The surface memory will be released when the last object referencing it is released. The primary surface, 3D surfaces, or implicitly created surfaces cannot be duplicated.

```
HRESULT DuplicateSurface(
    LPDIRECTDRAW lpDD,
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPLPDIRECTDRAWSURFACE lplpDupDDSurface)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpDDSurface
Points to the **DirectDrawSurface** structure that should be duplicated.

lplpDupDDSurface
Points to the **DirectDrawSurface** pointer which should point to the duplicate **DirectDrawSurface** structure that has just been created.

Return Values

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

EnumDisplayModes

Enumerate 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.

```
HRESULT EnumDisplayModes(
    LPDIRECTDRAW lpDD,
    DWORD dwFlags,
    LPDDSURFACEDESC lpDDSurfaceDesc,
    LPVOID lpContext,
    LPDDENUMMODESCALLBACK lpEnumCallback )
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwFlags
Not yet supported. Must be zero.

lpDDSurfaceDesc
Points to a DDSURFACEDESC structure to be checked against available modes. If NULL, all modes will be enumerated.

lpContext
Points to a caller defined structure that will be passed to each enumeration member.

lpEnumCallback
Points to the function the enumeration procedure will call everytime a match is found.

```
lpEnumCallback(
    LPDDMODEDESC lpDDModeDesc,
    LPVOID lpContext)
```

lpDDModeDesc
Points to the structure that contains the mode identifier, monitor frequency, and flags **DWORD** in addition to the included DDSURFACEDESC structure for a mode that provides the necessary functionality. This data is read-only.

lpContext
Points to the caller defined structure that is passed to the member every time it is invoked.

Return Values

DDENUMRET_OK	**Continue the enumeration**
DDENUMRET_CANCEL	**Stop the enumeration**

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

SetDisplayMode, **RestoreDisplayMode**

EnumSurfaces

Enumerate all of the existing or possible surfaces that meet the search criterion specified. If the DDENUMSURFACES_CANBECREATED flag is set, then this member 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, Release the surface after each enumeration.

```
HRESULT EnumSurfaces(
    LPDIRECTDRAW lpDD,
    DWORD dwFlags,
    LPDDSURFACEDESC lpDDSD,
    LPVOID lpContext,
    LPDDENUMSURFACESCALLBACK lpEnumCallback)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwFlags

DDENUMSURFACES_ALL
Enumerate all of the surfaces that meet the search criterion.

DDENUMSURFACES_MATCH
A search hit is a surface that matches the surface description.

DDENUMSURFACES_NOMATCH
A search hit is a surface that does not match the surface description.

DDENUMSURFACES_CANBECREATED
Enumerate the first surface that can be created which meets the search criterion.

DDENUMSURFACES_DOESEXIST
Enumerate the surfaces that already exist that meet the search criterion.

lplpDDSD
A pointer to a DDSURFACEDESC structure defining the surface of interest.

lpContext
Points to a caller-defined structure that will be passed to each enumeration member.

lpEnumCallback
Points to the function the enumeration procedure will call every time a match is found.

```
lpEnumCallback(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSURFACEDESC lpDDSurfaceDesc,
    LPVOID lpContext)
```

lpDDSurface
If *existing* surfaces are being enumerated (DDENUMSURFACES_DOESEXIST), then this pointer will point to the DirectDrawSurface currently being enumerated. If a *potential* surface is being enumerated (DDENUMSURFACES_CANBECREATED), then the value will be NULL.

lpDDSurfaceDesc
Points to the DDSURFACEDESC structure for the existing or potential surface that most closely matches the requested surface.

lpContext
Points to the caller defined structure that is passed to the member every time it is invoked.

Return Values

DDENUMRET_OK	**Continue the enumeration**
DDENUMRET_CANCEL	**Stop the enumeration**

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

FlipToGDISurface

Make the surface that GDI writes to the primary surface. This call is used at the end of a page flipping application to ensure that the video memory that GDI is writing to is visible to the user.

```
HRESULT FlipToGDISurface(
    LPDIRECTDRAW lpDD)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

Return Values

DD_OK	**DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT	**DDERR_NOTFOUND**

See Also

GetGDISurface

GetCaps

Fill in the *raw* (not remaining) capabilities of the device driver (the hardware) and/or the Hardware Emulation Layer (HEL).

```
HRESULT GetCaps(
    LPDIRECTDRAW lpDD,
    LPDDCAPS lpDDDriverCaps,
    LPDDCAPS lpDDHELCaps)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpDDDriverCaps
Points to a DDCAPS structure that will be filled in with the capabilities of the hardware (as reported by the device driver).

lpDDHELCaps
Points to a DDCAPS structure that will be filled in with the capabilities of the Hardware Emulation Layer (HEL).

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

DDCAPS

GetDisplayMode

Return the current display mode. An application *should not* save this mode in order to restore the display mode on clean up. The mode restoration on clean up should be done with RestoreDisplayMode to avoid mode setting conflicts that can arise in a multi-process environment.

```
HRESULT GetDisplayMode(
    LPDIRECTDRAW lpDD,
    LPDDSURFACEDESC lpDDSurfaceDesc)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpDDSurfaceDesc
Points to a DDSURFACEDESC structure that will be filled in with a description of the surface.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

RestoreDisplayMode, **EnumDisplayModes**

GetFourCCCodes

Get the FourCCCodes supported by the DirectDraw object. **GetFourCCCodes** can also be used to return the number of codes supported.

```
HRESULT GetFourCCCodes(
    LPDIRECTDRAW lpD,
    DWORD FAR *lpNumCodes,
    DWORD FAR *lpCodes)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpNumCodes
Points to a **DWORD** that contains the number of entries the **lpCodes** array can hold. If this number is too small to accomodate all the codes, it will be set to the required number and the **lpCodes** array will be filled with all that will fit.

lpCodes
Points to an array of **DWORD's** that will be filled in with the FourCC codes supported by this DirectDraw object. If **NULL** is passed, then **lpNumCodes** will be set to the number of supported FourCC codes and the member will return.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

GetGDISurface

Return the DirectDrawSurface object that currently represents the surface memory that GDI treats as the primary surface.

```
HRESULT GetGDISurface(
    LPDIRECTDRAW lpDD,
    LPDIRECTDRAWSURFACE FAR *lplpGDIDDSSurface)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lplpGDIDDSSurface
Points to a **DirectDrawSurface** pointer that will be made to point at the **DirectDrawSurface** object which is currently controlling GDI's primary surface memory.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOTFOUND**

See Also

FlipToGDISurface

GetMonitorFrequency

Return the frequency of the monitor being driven by the DirectDraw object.

```
HRESULT GetMonitorFrequency(
    LPDIRECTDRAW lpDD,
    LPDWORD lpdwFrequency)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpdwFrequency
Points to the **DWORD** that will be filled in with the monitor frequency. **Note**: 60Hz is returned 6000.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_UNSUPPORTED**

GetScanLine

Return the scan line that the monitor is currently updating to the display.

```
HRESULT GetScanLine(
    LPDIRECTDRAW lpDD,
    LPDWORD lpdwScanLine)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpdwScanLine
Points to the DWORD that will contain the scan line the display is currently on.

Return Values

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

See Also

GetVerticalBlankStatus, **WaitForVerticalBlank**

GetVerticalBlankStatus

Return the status of the vertical blank. It will set the passed **BOOL** to true if it is in the vertical blank and false otherwise. To synchronize with the vertical blank, consider using WaitForVerticalBlank.

```
HRESULT GetVerticalBlankStatus(
    LPDIRECTDRAW lpDD,
    LPBOOL lpbIsInVB)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

lpbIsInVB
Points to the **BOOL** that will be filled in with the status of the vertical blank.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

GetScanLine, **WaitForVerticalBlank**

Initialize

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

```
HRESULT Initialize(
    LPDIRECTDRAW lpDD,
    GUID FAR *lpGUID)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object

lpGUID
Points to the GUID used as the interface identifier.

Return Values

DDERR_ALREADYINITIALIZED

See Also

AddRef, **QueryInterface**, **Release**

QueryInterface

This member is part of the **IUnknown** interface inherited by **DirectDraw**. It is used to increase the reference count of the **DirectDraw** object. This is the member that applications use to determine whether the **DirectDraw** object supports additional interfaces that they may be interested in. An application can ask the **DirectDraw** object if it supports a particular **COM** interface and if it does the application may begin using that interface immediately. If the application does not want to use that interface it must call Release to free it. This member allows **DirectDraw** objects to be extended by Microsoft and third parties without breaking, or interfering with, each other's existing or future functionality.

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

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

riid
Points to a **UUID**. (Universally Unique Identifier).

obp
Points to a pointer that will be filled with the interface pointer if the query is successful.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

AddRef, **Initialize**, **Release**

Release

Decrease the reference count of the **DirectDraw** object. This member is part of the **IUnknown** interface inherited by **DirectDraw**. When the **DirectDraw** object is initially created its reference count is set to one. Each time **Release** is called by an application the **DirectDraw** object reduces the reference count by one. The **DirectDraw** object deallocates itself when its reference count goes to zero. The AddRef member is used to increase the reference count every time a new application binds to the **DirectDraw** object.

```
DWORD Release(
    LPDIRECTDRAW lpDD)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, **QueryInterface**, **Initialize**

RestoreDisplayMode

Reset the mode of the display device hardware for the primary surface to what it was before the SetDisplayMode member function was called to change it. Exclusive level access is required to use this member.

```
HRESULT RestoreDisplayMode(
    LPDIRECTDRAW lpDD)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_LOCKEDSURFACES**
DDERR_GENERIC	**DDERR_EXCLUSIVEMODENOTOWNED**

See Also

SetDisplayMode, **EnumDisplayModes**, **SetCooperativeLevel**

SetCooperativeLevel

This member determines the top-level behavior of the application. DDSCL_EXCLUSIVE level is needed to call functions that can have drastic performance consequences for other applications. In order to call Compact, change the display mode, or modify the behavior (e.g. flipping) of the primary surface, an application must have obtained exclusive level. If an application calls **SetCooperativeLevel** with DDSCL_EXCLUSIVE and DDSCL_FULLSCREEN, 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 | DDSCL_EXCLUSIVE. DDSCL_ALLOWMODEX cannot be used with DDSCL_NORMAL. If DDSCL_ALLOWMODEX is not specifed, EnumDisplayModes will not enumerate the ModeX modes, and SetDisplayMode will fail when a ModeX mode is requested. The set of supported display modes may change after using **SetCooperativeLevel**.

Because the ModeX modes are not supported by Windows, when in a ModeX mode you cannot Lock the primary surface, Blt to the primary surface, use GetDC on the primary surface, or use GDI with a screen DC. ModeX modes are indicated by the DDCAPS_MODEX flag in the DDSCAPS field of the DDSURFACEDESC structure returned by **Surface::GetCaps** and **EnumDisplayModes**.

```
HRESULT   SetCooperativeLevel(
    LPDIRECTDRAW lpDD,
    HWND hWnd,
    DWORD dwFlags)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

hWnd
Window handle used for the application.

dwFlags

DDSCL_ALLOWMODEX
Allow use of ModeX display modes.

DDSCL_ALLOWREBOOT
Allow CTRL_ALT_DEL to function while in fullscreen exclusive mode.

DDSCL_EXCLUSIVE
Application requests exclusive level.

DDSCL_FULLSCREEN
Exclusive mode owner will be responsible for the entire primary surface. GDI can be ignored.

DDSCL_NORMAL
Application will function as a regular Windows application.

DDSCL_NOWINDOWCHANGES
Don't allow DirectDraw to minimize/restore the application window on activation.

Return Values

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

See Also

SetDisplayMode, **Compact**, **EnumDisplayModes**

SetDisplayMode

Set the mode of the display device hardware. SetCooperativeLevel 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.

```
HRESULT SetDisplayMode(
    LPDIRECTDRAW lpDD,
    DWORD dwWidth,
    DWORD dwHeight,
    DWORD dwBpp)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwWidth
Width of the new mode.

dwHeight
Height of the new mode.

dwBpp
Bits per pixel of the new mode.

Return Values

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

See Also

RestoreDisplayMode, **EnumDisplayModes**, **SetCooperativeLevel**

WaitForVerticalBlank

This member is used to help the caller synchronize itself with the vertical blank interval. Depending on the option set with dwFlags, **WaitForVerticalBlank** will: 1) block until blank begins, 2) block until blank ends, or 3) trigger an event when the blank interval begins.

```
HRESULT WaitForVerticalBlank(
    LPDIRECTDRAW lpDD,
    DWORD dwFlags,
    HANDLE hEvent)
```

Parameters

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

dwFlags
Determines how to wait for the vertical blank.

DDWAITVB_BLOCKBEGIN
Return when the vertical blank interval begins.

DDWAITVB_BLOCKBEGINEVENT
Set up an event to trigger when the vertical blank begins. Not currently supported.

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

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

Return Values

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

See Also

GetVerticalBlankStatus, **GetScanLine**

Overview

The DirectDrawSurface object represents a two dimensional piece of memory that contains data. This data is in a form that is understood by the display hardware represented by the DirectDraw object which created the DirectDrawSurface object. A DirectDrawSurface object is created by the CreateSurface member function of the DirectDraw object. In general, although it is not required, the DirectDrawSurface object resides within the video RAM of the display card. Unless specifically stated when the DirectDrawSurface object is being created, 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 CPU.

DirectDrawSurface objects are aware of, and integrated with, the rest of the components composing the Windows display system. DirectDrawSurface objects can create HDCs for themselves that allow GDI functions to write to the surface memory represented by the DirectDrawSurface object. GDI thinks of these HDCs as Memory Dcs but the hardware accelerators are usually enabled for them if they are in video memory.

DIRECTDRAWSURFACE Member Implementation

The members **AddRef**, **QueryInterface**, and **Release** are from the standard **COM** interface **IUnknown,** which all **COM** interfaces inherit. These three members allow additional interfaces to be added to the DirectDrawSurface without affecting the functionality of the original interface.

Members

AddAttachedSurface
AddOverlayDirtyRect
AddRef
Blt
BltBatch
BltFast
DeleteAttachedSurfaces
EnumAttachedSurfaces
EnumOverlayZOrders
Flip
GetAttachedSurface
GetBltStatus
GetCaps
GetClipper
GetColorKey
GetDC
GetFlipStatus
GetOverlayPosition
GetPalette
GetPixelFormat
GetSurfaceDesc
Initialize
IsLost
Lock
QueryInterface
Release
ReleaseDC
Restore
SetClipper
SetColorKey
SetOverlayPosition
SetPalette
Unlock
UpdateOverlay
UpdateOverlayDisplay
UpdateOverlayZOrder

AddAttachedSurface

Attach a *Surface* to another *Surface*. Examples of possible Attachments include Zbuffers, AlphaChannels, and BackBuffers. Some Attachments automatically break other Attachments. For example, the 3DZBUFFER can only be Attached to one BACKBUFFER at a time. Attachment is not bi-directional, and a surface cannot be attached to itself. Emulated (system memory) surfaces 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 it is allowable to attach two surfaces of different types; for example, a flippable Zbuffer 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 Flip member will cycle one step through the surfaces.

```
HRESULT AddAttachedSurface(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDIRECTDRAWSURFACE lpDDSAttachedSurface)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDSAttachedSurface
Points to the **DIRECTDRAWSURFACE** that is to be attached.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_GENERIC	**DDERR_SURFACELOST**
DDERR_INVALIDPARAMS	**DDERR_SURFACEALREADYATTACHED**
DDERR_WASSTILLDRAWING	**DDERR_CANNOTATTACHSURFACE**

See Also

DeleteAttachedSurfaces, **EnumAttachedSurfaces**, **Flip**

AddOverlayDirtyRect

This member is used to build up the list of the rectangles that need to be updated the next time the UpdateOverlayDisplay member is called. This member is used for the software implementation. It is not needed if the overlay support is provided in hardware.

```
HRESULT AddOverlayDirtyRect(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPRECT lpRect)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpRect
Points to the **RECT** that needs to be updated.

Return Values

DD_OK	**DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT	**DDERR_UNSUPPORTED**

See Also

UpdateOverlayDisplay

AddRef

This member is part of the **IUnknown** interface inherited by **DIRECTDRAWSURFACE**. It is used to increase the reference count of the **DIRECTDRAWSURFACE** object. When the **DIRECTDRAWSURFACE** object is initially created its reference count is set to one. Each time a new application binds to the **DIRECTDRAWSURFACE** object, or a previously bound application binds to a different **COM** interface of the **DIRECTDRAWSURFACE** object, the reference count is increased by one. The **DIRECTDRAWSURFACE** object deallocates itself when its reference count goes to zero. The Release member is used to notify the **DIRECTDRAWSURFACE** object that an application is no longer bound to it.

```
DWORD AddRef(
    LPDIRECTDRAWSURFACE lpDDSurface)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

CreateSurface, **Initialize**, **QueryInterface**, **Release**

Blt

Perform a bit block transfer. This member is capable of synchronous or asynchronous blits, either video memory to video memory, video memory to system memory, system memory to video 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/shrinking will be performed if the source and destination rectangles are not the same size.

Normally, **Blt** will return immediately with an error if the blitter is busy and the blit could not be set up. The DDBLT_WAIT flag can be used to alter this behavior such that **Blt** will wait until the blit can be set up, or another error occurs, before returning.

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the destination of the blit operation.

lpDestRect
Points to a **RECT** structure that defines the upper left and lower right points of the rectangle on the destination surface which is to be blted to.

lpDDSrcSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the source for the blit operation.

lpSrcRect
Points to a **RECT** structure that defines the upper left and lower right points of the rectangle on the source surface which is to be blted from.

dwFlags

DDBLT_ALPHADEST
Use 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
Use the dwConstAlphaDest field 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
Use the lpDDSAlphaDest field in the DDBLTFX structure as the alpha channel for the destination for this blit.

DDBLT_ALPHAEDGEBLEND
Use the dwAlphaEdgeBlend field in the DDBLTFX structure as the alpha channel for the edges of the image that border the color key colors.

DDBLT_ALPHASRC
Use 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

Use the dwConstAlphaSrc field 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
Use the lpDDSAlphaSrc field in the DDBLTFX structure as the alpha channel for the source for this blit.

DDBLT_ASYNC
Do this blit asynchronously through the FIFO in the order received. If there is no room in the hardware FIFO fail the call.

DDBLT_COLORFILL
Uses the dwFillColor field in the DDBLTFX structure as the RGB color to fill the destination rectangle on the destination surface with.

DDBLT_DDFX
Uses the dwDDFX field in the DDBLTFX structure to specify the effects to use for the blit.

DDBLT_DDROPS
Uses the dwDDROPS field in the DDBLTFX structure to specify the ROPS that are not part of the Win32 API.

DDBLT_KEYDEST
Use the color key associated with the destination surface.

DDBLT_KEYDESTOVERRIDE
Use the dckDestColorkey field in the DDBLTFX structure as the color key for the destination surface.

DDBLT_KEYSRC
Use the color key associated with the source surface.

DDBLT_KEYSRCOVERRIDE
Use the dckSrcColorkey field in the DDBLTFX structure as the color key for the source surface.

DDBLT_ROP
Use the dwROP field in the DDBLTFX structure for the raster operation for this blit. These ROPs are the same as the ones defined in the Win32 API.

DDBLT_ROTATIONANGLE
Use the dwRotationAngle field in the DDBLTFX structure as the angle (specified in 1/100th of a degree) to rotate the surface.

DDBLT_WAIT
Do not return immediately with the DDERR_WASSTILLDRAWING message if the bltter is busy -- wait until the Blt can be set up or another error occurs.

DDBLT_ZBUFFER
Z-buffered blit using the z-buffers attached to the source and destination surfaces and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer opcode.

DDBLT_ZBUFFERDESTCONSTOVERRIDE
Z-buffered blit using the dwConstDest Zfield and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the destination.

DDBLT_ZBUFFERDESTOVERRIDE
Z-buffered blit using the lpDDSDestZBuffer field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the destination.

DDBLT_ZBUFFERSRCCONSTOVERRIDE
Z-buffered blit using the dwConstSrcZ field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the source.

DDBLT_ZBUFFERSRCOVERRIDE
Z-buffered blit using the lpDDSSrcZBuffer field and the dwZBufferOpCode field in the DDBLTFX structure as the z-buffer and z-buffer opcode respectively for the source.

lpDDBltFx

See DDBLTFX structure

DDBLTFX_ALPHA **DDBLTFX_MIRRORLEFTRIGHT**
DDBLTFX_MIRRORUPDOWN **DDBLTFX_NOTEARING**
DDBLTFX_ROTATE180 **DDBLTFX_ROTATE270**
DDBLTFX_ROTATE90

Return Values

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

BltBatch

Perform a sequence of Blt operations from several sources to a single destination.

```
HRESULT BltBatch(
    LPDIRECTDRAWSURFACE lpDDDestSurface,
    LPDDBLTBATCH lpDDBltBatch,
    DWORD dwCount,
    DWORD dwFlags)
```

Parameters

lpDDDestSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the destination of the blit operations.

lpDDBltBatch
Points to the first DDBLTBATCH structure defining the parameters for the blit operations.

dwCount
The number of blit operations to be performed.

dwFlags
Not currently used.

Return Values

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

BltFast

Perform a source copy blit or transparent blit using a source or destination color key. **BltFast** always attempts to perform an asynchronous blit if the hardware supports this. It only works on video memory surfaces and cannot clip. The software implementation of **BltFast** is 10% faster than Blt. There is no speed difference if the video hardware is being used.

Normally, **BltFast** will return immediately with an error if the blitter is busy and the blit could not be set up. The DDBLTFAST_WAIT flag can be used to alter this behavior such that **BltFast** will wait until the blit can be set up, or another error occurs, before returning.

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the destination of the blit operation.

dwX
X coordinate on destination surface to blit to.

dwY
Y coordinate on destination surface to blit to.

lpDDSrcSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the source for the blit operation.

lpSrcRect
Points to a **RECT** structure that defines the upper left and lower right points of the rectangle on the source surface which is to be blted from.

dwTrans
Specify the type of transfer.

DDBLTFAST_DESTCOLORKEY
Transparent blit using the destination's color key.

DDBLTFAST_NOCOLORKEY
Normal copy blit -- no transparency.

DDBLTFAST_SRCCOLORKEY
Transparent blit using the source's color key.

DDBLTFAST_WAIT
Do not return immediately with the DDERR_WASSTILLDRAWING message if the bltter is busy -- wait until the Blt can be set up or another error occurs.

Return Values

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

DeleteAttachedSurfaces

Detache two attached surfaces. The detached surface is not released. If NULL is passed as the surface to be detached, all attached surfaces will be detached. Implicit attachments (those formed by DirectDraw, rather than AddAttachedSurfaces) cannot be detached. Detaching surfaces from a flippable chain can change other surfaces in the chain. If a FRONTBUFFER is detached from a flippable chain, the next surface in the chain becomes the FRONTBUFFER and the surface following it becomes the BACKBUFFER. If a BACKBUFFER is detached from a chain, the following surface becomes a BACKBUFFER. 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 flippable chain is destroyed and both surfaces return to their previous designations.

```
HRESULT DeleteAttachedSurfaces(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags,
    LPDIRECTDRAWSURFACE lpDDSAttachedSurface)
```

Parameter

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags
0 is the only valid value.

lpDDSAttachedSurface
Points to the **DIRECTDRAWSURACE** structure which is to be detached. If NULL is passed, all attached surfaces will be detached.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_SURFACELOST	**DDERR_SURFACENOTATTACHED**
DDERR_INVALIDPARAMS	**DDERR_CANNOTDETACHSURFACE**

See Also

Flip

EnumAttachedSurfaces

Enumerate all the surfaces attached to a given surface.

```
HRESULT EnumAttachedSurfaces(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPVOID lpContext,
    LPDDENUMSURFACESCALLBACK lpEnumSurfacesCallback )
```

Parameter

lpDDSurface

Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpContext

Points to the caller defined structure that is passed to the enumeration member every time it is called.

lpEnumSurfacesCallback

Points to the callback function that will be called for each surface that is attached to this surface.

```
lpEnumSurfacesCallback(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSURFACEDESC lpDDSurfaceDesc,
    LPVOID lpContext)
```

lpDDSurface

Points to the surface that is attached to this surface.

lpDDSurfaceDesc

Points to a DDSURFACEDESC structure that describes the attached surface.

lpContext

Points to the user defined context that was specified by the user.

Return Value

DDENUMRET_OK	**Continue the enumeration**
DDENUMRET_CANCEL	**Stop the enumeration**

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**

EnumOverlayZOrders

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

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags

DDENUMOVERLAYZ_BACKTOFRONT
Enumerate overlays back to front.

DDENUMOVERLAYZ_FRONTTOBACK
Enumerate overlays front to back.

lpContext
Points to the user defined context that will be passed to the callback for each overlay surface.

lpfnCallback
Points to the callback function that will be called for each overlay that is being overlayed on this surface.

```
lpfnCallback(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPVOID lpContext)
```

lpDDSurface
Points to the surface that is being overlayed on this surface.

lpContext
Points to the user defined context that was specified by the user.

Return Value

DDENUMRET_OK	**Continue the enumeration**
DDENUMRET_CANCEL	**Stop the enumeration**

Return Values

DD_OK
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Flip

Make the surface memory associated with the DDSCAPS_BACKBUFFER surface become associated with the FRONTBUFFER surface. This member can only be called by a surface that has the DDSCAPS_FLIP and DDSCAPS_FRONTBUFFER bits set. The video memory previously associated with the front buffer is associated with the back buffer. If there is more than one back buffer, then a ring is formed and the surface memory buffers cycle one step through it every time **Flip** is invoked.

The TargetOverride parameter is used in rare cases where the BACKBUFFER is not the buffer that should become the FRONTBUFFER. Normally it is NULL.

Flip will always be synchronized with the vertical blank.

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDSurfaceTargetOverride
Points to the **DIRECTDRAWSURFACE** structure which is to be flipped to. The default for this parameter is NULL, in which case 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
Normally, if the Flip cannot be set up because the state of the video hardware is not appropriate, the error WASSTILLDRAWING will be returned immediately and no flip will occur. Setting this flag will cause Flip to continue trying if it receives the WASSTILLDRAWING error from the HAL. Flip will not return until the flipping operation has been successfully set up or another error, such as SURFACEBUSY, has occurred.

Return Values

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

See Also

GetFlipStatus

GetAttachedSurface

Find the attached surface that has the specified capabilities. Attachments are used to connect multiple **DIRECTDRAWSURFACE** objects into Complex Structures like the ones needed to support 3D Page Flipping with Z Buffers. **GetAttachedSurface** will fail if there is more than one surface attached which matches the capabilities requested. In this case the application must use EnumAttachedSurfaces to obtain the non-unique attached surfaces.

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDSCaps
Points to a DDCAPS structure that contains the hardware capabilities of the surface.

lplpDDAttachedSurface
Points to a pointer that will be filled with the address of the **DIRECTDRAWSURFACE** that is attached to the requesting surface and has the appropriate capabilities.

Return Values

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

GetBltStatus

Return the blitter status. This member returns OK if a blitter is present, WASSTILLDRAWING if the bltter is busy, or NOBLTHW if there is no blitter.

```
HRESULT GetBltStatus(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags)
```

lpDDSurface

Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags

DDGBS_CANBLT

Can a blit occur involving this surface now? Returns OK if the blit can be completed.

DDGBS_ISBLTDONE

Is the blit done? Returns OK if the last blit on this surface has completed.

Return Values

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

GetCaps

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

```
HRESULT GetCaps(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSCAPS lpDDSCaps)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDCaps
Points to a DDCAPS structure that will be filled in with the hardware capabilities of the surface.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

GetClipper

Return the DirectDrawClipper associated with this surface. It returns an error if there is no DirectDrawClipper associated.

```
HRESULT GetClipper(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDIRECTDRAWCLIPPER FAR * lplpDDClipper)
```

Parameters

lpDDSurface
Points to the **DirectDrawSurface** structure representing the DirectDrawSurface.

lplpDDClipper
Points to a pointer to the DirectDrawClipper associated with the surface.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOCLIPPERATTACHED**

See Also

SetClipper

GetColorKey

Return the color key value for the **DIRECTDRAWSURFACE** object.

```
HRESULT GetColorKey(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags,
    LPDDCOLORKEY lpDDColorKey)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags
Determines which color key is being requested.

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

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

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

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

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

lpDDColorKey
Points to the **DDCOLORKEY** structure that will be filled in with the current values for the specified color key for the **DIRECTDRAWSURFACE** object.

Return Values

DD_OK
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACELOST
DDERR_UNSUPPORTED
DDERR_NOCOLORKEYHW
DDERR_NOCOLORKEY

See Also

SetColorKey

GetDC

Create a GDI compatible hDC for the surface. *It uses an internal version of the* Lock *member to lock the surface, and the surface will remain locked until DirectDraw's* ReleaseDC *member is called*. See the description of the Lock member for more information on this behavior.

```
HRESULT GetDC(
    LPDIRECTDRAWSURFACE lpDirectDrawSurface,
    HDCFAR *lphDC)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lphDC
Points to the hDC returned.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**
DDERR_WASSTILLDRAWING	**DDERR_GENERIC**
DDERR_UNSUPPORTED	

See Also

ReleaseDC, **Lock**

GetFlipStatus

This member returns OK if the surface that it is called on has finished its flipping process, otherwise it returns WASSTILLDRAWING.

```
HRESULT GetFlipStatus(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags)
```

Parameters

lpDDSurface
Points to the DIRECTDRAWSURFACE structure representing the DirectDrawSurface.

dwFlags

DDGFS_CANFLIP
Can this surface be flipped now? Returns OK if the flip can be completed.

DDGFS_ISFLIPDONE
Is the flip done? Returns OK if the last flip on this surface has completed.

Return Values

DD_OK
DDERR_INVALIDOBJECT
DDERR_UNSUPPORTED
DDERR_SURFACEBUSY
DDERR_INVALIDPARAMS
DDERR_SURFACELOST
DDERR_WASSTILLDRAWING

See Also

Flip

GetOverlayPosition

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

```
HRESULT GetOverlayPosition(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPLONG lplX,
    LPLONG lplY)
```

Parameters

lpDirectDrawSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lplX
Points to the X display coordinate.

lplY
Points to the Y display coordinate.

Return Values

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

See Also

SetOverlayPosition, **UpdateOverlay**

GetPalette

Return the **DIRECTDRAWPALETTE** structure associated with this surface. If no palette has been explicitly associated with this surface then it returns **NULL** for the associated palette, unless this is the primary surface or a back buffer to the primary surface, in which case it returns a pointer to the system palette if the primary surface is in 8bpp mode.

```
HRESULT GetPalette(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPLPDIRECTDRAWPALETTE lplpDDPalette)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lplpDDPalette
Points to 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 there is no **DIRECTDRAWPALETTE** associated with this surface.

Return Values

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

See Also

SetPalette

GetPixelFormat

Return the color and pixel format of the surface.

```
HRESULT GetPixelFormat(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDPIXELFORMAT lpDDPixelFormat)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDPixelFormat
Points to the DDPIXELFORMAT structure which will be filled in with a detailed description of the current pixel and color space format of the surface.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

GetSurfaceDesc

Return a DDSURFACEDESCstructure describing the surface in its current condition.

```
HRESULT GetSurfaceDesc(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDDSURFACEDESC lpDDSurfaceDesc)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

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

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

DDSURFACEDESC

Initialize

Initialize a DirectDrawSurface. This member is provided for compliance with the Common Object Model (COM) protocol. Since the DirectDrawSurface object is initialized when it is created, calling this member will always result in the ALREADYINITIALIZED return value.

```
HRESULT Initialize(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDIRECTDRAW lpDD,
    LPDDSURFACEDESC lpDDSurfaceDesc )
```

Parameters

lpDDSurface
Points to the **DirectDrawSurface** structure representing the DirectDrawSurface.

lpDD
Points to the **DirectDraw** structure representing the DirectDraw object.

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

Return Values

DDERR_ALREADYINITIALIZED

See Also

AddRef, **QueryInterface**, **Release**

IsLost

Determine if the surface memory associated with a DirectDrawSurface has been freed. If the memory has not been freed, this member will return OK. The Restore member can be used to reallocate surface memory. When a DirectDrawSurface object loses its surface memory, most members will return 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 video card.

```
HRESULT IsLost(
    LPDIRECTDRAWSURFACE lpDDSurface)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_SURFACELOST**

See Also

Restore

Lock

Obtain a valid pointer to the surface memory. DirectDraw relies on the application calling Unlock. It is illegal behavior to Blt from a region of a surface that is locked.

An application should call the **Lock** member with a **RECT** structure specifying the rectangle on the surface that the application wants access to. If the application calls **Lock** with a **NULL RECT** then the application is assumed to be requesting exclusive access to the entire piece of surface memory.

The **Lock** member fills in a DDSURFACEDESCstructure with the information needed by the application to access the surface memory. This information includes the stride (or pitch) and the pixel format of the surface if it is different from the pixel format of the primary surface. The **DDSURFACEDESC** structure also contains a pointer to the surface memory. Since it is possible to call **Lock** multiple times for the same *Surface* with different destination rectangles, this pointer is used to tie the **Lock** and **Unlock** calls together.

Normally, **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 used to alter this behavior.

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

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDestRect
Points to a **RECT** structure identyfing the region of **surface** that is being locked.

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

dwFlags

DDLOCK_SURFACEMEMORYPTR
The default. Set to indicate that Lock should return a valid memory pointer to the top of the specified rectangle. If no rectangle is specified then a pointer to the top of the surface is returned.

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

DDLOCK_WAIT
Normally, if a lock cannot be obtained because a Blt is in progress, a WASSTILLDRAWING error will be returned immediately. If this flag is set, however, **Lock** will retry until a lock is obtained or another error, such as SURFACEBUSY, occurs.

hEvent

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

Return Values

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

See Also

Unlock, **GetDC**, **ReleaseDC**

QueryInterface

This member is part of the **IUnknown** interface inherited by **DIRECTDRAWSURFACE**. It is used to increase the reference count of the **DIRECTDRAWSURFACE** object. This is the member that applications use to determine whether the **DIRECTDRAWSURFACE** object supports additional interfaces that they may be interested in. An application can ask the **DIRECTDRAWSURFACE** object if it supports a particular **COM** interface, and if it does, the application may begin using that interface immediately. If the application does not want to use that interface it must call Release to free it. This member allows **DIRECTDRAWSURFACE** objects to be extended by Microsoft and third parties without breaking, or interfering with, each other's existing or future functionality.

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

riid
Points to a **UUID**. (Universally Unique Identifier)

ppvObj
Points to a pointer that will be filled with the interface pointer if the query is successful.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_OUTOFMEMORY**

See Also

AddRef, **Initialize**, **Release**

Release

This member is part of the **IUnknown** interface inherited by **DIRECTDRAWSURFACE**. It is used to decrease the reference count of the **DIRECTDRAWSURFACE** object. When the **DIRECTDRAWSURFACE** object is initially created, its reference count is set to one. Each time **Release** is called by an application the **DIRECTDRAWSURFACE** object reduces the reference count by one. The **DIRECTDRAWSURFACE** object deallocates itself when its reference count goes to zero. The AddRef member is used to increase the reference count every time a new application binds to the **DIRECTDRAWSURFACE** object.

Implicit surfaces created by DirectDraw to satisfy "complex" requests should be released by releasing the front buffer of the complex structure. Releasing an implicit surface directly is not allowed. For example, a complex flipping structure creating with the DDSCAPS_OFFSCREENPLAIN, DDSCAPS_COMPLEX, and DDSCAPS_FLIP flags, with 2 back buffers, is released with a single call to **Release**, passing the front buffer surface, not individual calls for each surface.

```
DWORD Release(
    LPDIRECTDRAWSURFACE lpDDSurface)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, **Initialize**, **QueryInterface**

ReleaseDC

Release the hDC previously obtained with GetDC. It also Unlocks the surface previously Locked when GetDC was called.

```
HRESULT ReleaseDC(
    LPDIRECTDRAWSURFACE lpDDSurface,
    HDC hDC)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

hDC
hDC previously obtained by **GetDC**.

Return Values

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

See Also

GetDC

Restore

Restore a surface that has been "lost" -- the surface memory associated with the DirectDrawSurface object has been freed. Surfaces can be lost because the mode of the display card was changed or because an application received exclusive access to the display card and freed all of the surface memory currently allocated on the video card. When a DirectDrawSurface object loses its surface memory, many members will return SURFACELOST and perform no other function. **Restore** will reallocate surface memory and reattach it to the DirectDrawSurface object.

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

```
HRESULT Restore(
    LPDIRECTDRAWSURFACE lpDDSurface)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

Return Values

DD_OK
DDERR_INVALIDPARAMS
DDERR_INVALIDOBJECT
DDERR_OUTOFMEMORY
DDERR_GENERIC
DDERR_UNSUPPORTED
DDERR_IMPLICITLYCREATED
DDERR_WRONGMODE
DDERR_NOEXCLUSIVEMODE
DDERR_INCOMPATIBLEPRIMARY

See Also

IsLost, **AddAttachedSurface**

SetClipper

Attaches a DirectDrawClipper to a DirectDrawSurface. This function is primarily used by surfaces that are being overlayed on or blted to the primary surface, but it can be used on any surface. Once a DirectDrawClipper has been attached, and a clip list associated with it, it will be used for Blt, BltBatch, and UpdateOverlay operations involving the parent DirectDrawSurface. This member can also be used to detach a DirectDrawSurface's current Clipper.

```
HRESULT SetClipper(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDIRECTDRAWCLIPPER lpDDClipper)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDirectDrawClipper
Either NULL, or points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper that will be attached to the DirectDrawSurface. If NULL, the current clipper will be detached.

Return Values

DD_OK	**DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT	**DDERR_NOCLIPPERATTACHED**

See Also

GetClipper

SetColorKey

Set the color key value for the **DIRECTDRAWSURFACE** object if the hardware supports color keys on a per surface basis.

```
HRESULT SetColorKey(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags,
    LPDDCOLORKEY lpDDColorKey)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags
Determines which color key is being requested.

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

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

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

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

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

lpDDColorKey
Points to the DDCOLORKEY structure which has the new color key values for the **DIRECTDRAWSURFACE** object.

Return Values

DD_OK
DDERR_NOOVERLAYHW **DDERR_COLORKEYDRIVERWIDE**
DDERR_NODESTCLRKEYHW **DDERR_NOSRCCLRKEYHW**
DDERR_INVALIDPARAMS **DDERR_INVALIDOBJECT**
DDERR_SURFACELOST **DDERR_UNSUPPORTED**
DDERR_WASSTILLDRAWING **DDERR_GENERIC**
DDERR_NOTAOVERLAYSURFACE

See Also

GetColorKey

SetOverlayPosition

Change the display coordinates of an overlay surface.

```
HRESULT SetOverlayPosition(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LONG lX,
    LONG lY)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lX
New X display coordinate.

lY
New Y display coordinate.

Return Values

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

See Also

GetOverlayPosition, **UpdateOverlay**

SetPalette

Attach the **DIRECTDRAWPALETTE** specified to a *Surface*. The *Surface* will use this *Palette* for all subsequent operations. The palette change takes place immediately, without regard to refresh timing.

```
HRESULT SetPalette(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPDIRECTDRAWPALETTE lpDDPalette)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpDDPalette
Pointer to the **DIRECTDRAWPALETTE** structure that this *Surface* should use for future operations.

Return Values

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

See Also

GetPalette, **CreatePalette**

Unlock

Notify DirectDraw that the direct surface manipulations are complete.

```
HRESULT Unlock(
    LPDIRECTDRAWSURFACE lpDDSurface,
    LPVOID lpSurfaceData)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

lpSurfaceData
This is the pointer returned by Lock. Since it is possible to call **Lock** multiple times for the same *Surface* with different destination rectangles, this pointer is used to tie the **Lock** and **Unlock** calls together.

Return Values

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

See also

Lock

UpdateOverlay

Reposition and/or modify the visual attributes of an overlay surface. These surfaces must have the **DDSCAPS_OVERLAY** bit set.

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

Parameters

lpDDSrcSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the source for the overlay operation.

dwSrcRect
Points to a **RECT** structure that defines the **X**, **Y**, **Width**, and **Height** of the region on the source surface which is being used as the overlay.

lpDDDestSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface. This is the surface that is being overlayed.

lpDestRect
Points to a **RECT** structure that defines the **X**, **Y**, **Width**, and **Height** of the region on the destination surface which the overlay should be moved to.

dwFlags

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

DDOVER_ALPHADESTCONSTOVERRIDE
Use the dwConstAlphaDest field 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.

DDOVER_ALPHADESTSURFACEOVERRIDE
Use the lpDDSAlphaDest field in the DDOVERLAYFX structure as the alpha channel destination for this overlay.

DDOVER_ALPHAEDGEBLEND
Use the dwAlphaEdgeBlend field in the DDOVERLAYFX structure as the alpha channel for the edges of the image that border the color key colors.

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

DDOVER_ALPHASRCCONSTOVERRIDE
Use the dwConstAlphaSrc field 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
Use the lpDDSAlphaSrc field in the DDOVERLAYFX structure as the alpha channel

source for this overlay.

DDOVER_HARDWAREONLY
This overlay must be done in hardware. If neither this flag nor the DDOVER_HARDWAREONLY flag is set DirectDraw tries to use hardware first and then falls back to software emulation if possible. NOTE: May not be defined in the current verision of DDRAW.H

DDOVER_HIDE
Turn this overlay off.

DDOVER_KEYDEST
Use the color key associated with the destination surface.

DDOVER_KEYDESTOVERRIDE
Use the dckDestColorkey field in the DDOVERLAYFX structure as the color key for the destination surface.

DDOVER_KEYSRC
Use the color key associated with the source surface.

DDOVER_KEYSRCOVERRIDE
Use the dckSrcColorkey field in the DDOVERLAYFX structure as the color key for the source surface.

DDOVER_SHOW
Turn this overlay on.

DDOVER_SOFTWAREONLY
This overlay must be done in software. If neither this flag nor the DDOVER_HARDWAREONLY flag is set DirectDraw tries to use hardware first and then falls back to software emulation if possible.

DDOVER_ZORDER
Use the dwZOrderFlags field in the DDOVERLAYFX structure as the z order for the display of this overlay. The lpDDSRelative field will be used if the dwZOrderFlags field is set to either DDOVERZ_INSERTINBACKOF or DDOVERZ_INSERTINFRONTOF.

lpDDOverlayFx
See **DDOVERLAYFX structure**

Return Values

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

UpdateOverlayDisplay

Repaint the rectangles in the dirty rectangle lists of all active overlays. The dirty rectangle list is cleared. This function is for software emulation only -- it does nothing if the hardware supports overlays.

```
HRESULT UpdateOverlayDisplay(
    LPDIRECTDRAWSURFACE lpDDSurface,
    DWORD dwFlags)
```

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags
Specify type of update to perform.

DDOVER_REFRESHDIRTYRECTS
Update the overlay display using the list of dirty rectangles previously constructed for this destination. The dirty rectangle list is cleared after this.

DDOVER_REFRESHALL
Ignore the dirty rectangle list and update the overlay display completely. The dirty rectangle list is cleared after this.

Return Values

DD_OK	**DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT	**DDERR_UNSUPPORTED**

See Also

AddOverlayDirtyRect

UpdateOverlayZOrder

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

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

Parameters

lpDDSurface
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface.

dwFlags

DDOVERZ_INSERTINBACKOF
Insert this overlay in the overlay chain in back of the overlay specified as the reference overlay.

DDOVERZ_INSERTINFRONTOF
Insert this overlay in the overlay chain in front of the overlay specified as the reference overlay.

DDOVERZ_MOVEBACKWARD
Move this overlay one overlay position backward

DDOVERZ_MOVEFORWARD
Move this overlay one overlay position forward.

DDOVERZ_SENDTOBACK
Move this overlay to the back of the overlay chain

DDOVERZ_SENDTOFRONT
Move this overlay to the front of the overlay chain.

lpDDSReference
Points to the **DIRECTDRAWSURFACE** structure representing the DirectDrawSurface that should be used as a relative position in the overlay chain. This parameter is needed only for **DDOVERZ_INSERTINBACKOF** and **DDOVERZ_INSERTINFRONTOF**.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOTAOVERLAYSURFACE**

See Also

EnumOverlayZOrders

Overview

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

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

DirectDrawPalette objects are usually attached to DirectDrawSurface objects.

Palette animation is straightforward using DirectDrawPalette objects. There are two approaches. The first involves simply changing the palette entries that correspond to the colors which need to be animated. This can be done with a single call to the **SetEntries** member function. The second one requires two DirectDrawPalette objects. The animation is performed by attaching first one and then the other to the DirectDrawSurface. This can be done using the SetPalette member function of the DirectDrawSurface objects.

DIRECTDRAWPALETTE Member Implementation

The members **AddRef**, **QueryInterface**, and **Release** are from the standard **COM** interface **IUnknown** which all **COM** interfaces inherit. These three members allow additional interfaces to be added to the DirectDrawPalette without effecting the functionality of the original interface.

Members

AddRef
GetCaps
GetEntries
Initialize
QueryInterface
Release
SetEntries

AddRef

This member increases the reference count of a **DirectDrawPalette** object previously created by the CreatePalette member function of a **DirectDraw** object.

```
DWORD AddRef(
    LPDIRECTDRAWPALETTE lpDDPalette)
```

Parameters

lpDDPalette
Points to the **DIRECTDRAWPALETTE** structure which was returned to the application when the DirectDrawPalette was created.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

Initialize, **QueryInterface**, **Release**

GetCaps

Return the capabilities of this palette object.

```
HRESULT GetCaps(
    LPDIRECTDRAWPALETTE lpDDPalette,
    LPDWORD lpdwCaps)
```

Parameters

lpDDPalette

Points to the **DIRECTDRAWPALETTE** structure which was returned to the application when the DirectDrawPalette was created.

lpdwCaps

Flags for dwPalCaps.

DDPCAPS_4BIT	**DDPCAPS_8BITENTRIES**
DDPCAPS_8BIT	**DDPCAPS_ALLOW256**
DDPCAPS_PRIMARYSURFACE	**DDPCAPS_PRIMARYSURFACELEFT**
DDPCAPS_VSYNC	

Return Values

DD_OK	**DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT	

GetEntries

Query palette values from a DirectDrawPalette.

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

Parameters

lpDDPalette
Points to the **DIRECTDRAWPALETTE** structure which was returned to the application when the DirectDraw member CreatePalette was called.

dwFlags
Not used at this time. Must be zero.

dwBase
This is the start of the entries that should be retrieved sequentially.

dwNumEntries
How many palette entries does **lpPixelEntryRequests** have room for. The colors of each palette entry will be returned in sequence starting from **dwStartingEntry** and proceeding through **dwCount** - 1.

lpPaletteEntries
A pointer to the palette entries. The palette entries are one byte each if the **DDPCAPS_8BITENTRIES** field is set and four bytes otherwise. Each field is a color description.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_NOTPALETTIZED**

See Also

SetEntries

Initialize

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

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

Parameters

lpDDPalette
Points to the **DIRECTDRAWPALETTE** structure which was returned to the application when the DirectDraw member **CreatePalette** was called.

lpDD
Points to the **DIRECTDRAWSTUCTURE** representing the DirectDraw object.

dwFlags
Not used.

lpDDColorTable
Not used.

Return Values

DDERR_ALREADYINITIALIZED

See Also

AddRef, **QueryInterface**, **Initialize**

QueryInterface

This member is part of the **IUnknown** interface inherited by the DirectDrawPalette. It is used to increase the reference count of the DirectDrawPalette object. This is the member that applications use to determine whether the DirectDrawPalette object supports additional interfaces that they may be interested in. An application can ask the DirectDrawPalette object if it supports a particular COM interface, and if it does, the application may begin using that interface immediately. If the application does not want to use that interface it must call Release to free it. This member allows DirectDrawPalette objects to be extended by Microsoft and third parties without breaking, or interfering with, each other's existing or future functionality.

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

Parameters

lpDDPalette
Points to the **DIRECTDRAWPALETTE** structure which was returned to the application when the DirectDrawPalette was created.

riid
Points to the GUID used as the interface identifier.

ppvObj
Points to a pointer which will receive the interface pointer if the request is successful.

Return Values

DD_OK **DDERR_INVALIDPARAMS**
DDERR_INVALIDOBJECT

See Also

AddRef, **Initialize**, **Release**

Release

This function reduces the interface reference count on the **DIRECTDRAWPALETTE** created by and returned from the **DirectDraw** member CreatePalette. When the reference count reaches zero, the object will be freed.

```
DWORD Release(
    LPDIRECTDRAWPALETTE lpDDPalette)
```

Parameters

lpDDPalette
Points to the **DIRECTDRAWPALETTE** structure which was returned to the client when the DirectDraw member **CreatePalette** was called to create the DirectDrawPalette.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, **Initialize**, **QueryInterface**

SetEntries

Change entries in a DirectDrawPalette. The changes will be performed immediately. The palette must be attached to a surface using the SetPalette member before **SetEntries** can be used.

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

Parameters

lpDDPalette
This is the pointer to the **DIRECTDRAWPALETTE** structure which was returned to the client when the DirectDraw member **CreatePalette** was called to create the target palette.

dwFlags
Not currently used. Must be zero.

dwStartingEntry
The first entry to be set.

dwCount
The number of palette entries to be changed.

lpPixelEntryReplacements
The palette entries are one byte each if the **DDPCAPS_8BITENTRIES** field is set and four bytes otherwise. Each field is a color description.

Return Values

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

See Also

GetEntries, **SetPalette**

Overview

Cliplists 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's clip list with the clip list for the window as it changes.

DIRECTDRAWCLIPPER Member Implementation

The members **AddRef**, **QueryInterface**, and **Release** are from the standard **COM** interface **IUnknown** which all **COM** interfaces inherit. These three members allow additional interfaces to be added to the DirectDrawClipper without effecting the functionality of the original interface.

Members

AddRef
GetClipList
GetHWnd
Initialize
IsClipListChanged
QueryInterface
Release
SetClipList
SetHWnd

AddRef

This member is part of the **IUnknown** interface inherited by the **DirectDrawClipper**. It is used to increase the reference count of the **DirectDrawClipper** object. When the **DirectDrawClipper** object is initially created, its reference count is set to one. Each time a new application binds to the **DirectDrawClipper** object, or a previously bound application binds to a different **COM** interface of the **DirectDraw** object, the reference count is increased by one. The **DirectDrawClipper** object deallocates itself when its reference count goes to zero. The Release member is used to notify the **DirectDraw** object that an application is no longer bound to it.

```
DWORD AddRef(
    LPDIRECTDRAWCLIPPER lpDDClipper )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure which was returned to the application when the DirectDrawClipper was created using the CreateClipper member of the DirectDraw object.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, CreateClipper, Initialize, QueryInterface, Release

GetClipList

Return a copy of the clip list associated with a DirectDrawClipper. A subset of the clip list can be selected by passing a rectangle which will be used to clip the clip list.

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

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper.

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

lpClipList
Points to a RGNDATA structure which will contain the resulting copy of the clip list.

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

Return Values

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

See Also

SetClipList

GetHWnd

Returns the hWnd previously associated with this DirectDrawClipper using the SetHWnd member.

```
HRESULT GetHWnd(
    LPDIRECTDRAWCLIPPER lpDDClipper,
    HWND FAR *lphWnd )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper.

lphWnd
Points to the hWnd previously associated with this DirectDrawClipper using the **SetHWnd** member.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

SetHwnd

Initialize

Initialize a DirectDrawClipper. This member is provided for compliance with the Common Object Model (COM) protocol. Since the DirectDrawClipper object is initialized when it is created, calling this member will always result in the ALREADYINITIALIZED return value.

```
HRESULT Initialize(
    LPDIRECTDRAWCLIPPER lpDDClipper,
    LPDIRECTDRAW lpDD,
    DWORD dwFlags )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure which was returned to the application when the DirectDraw member **CreateClipper** was called.

lpDD
Points to the **DIRECTDRAW** structure representing the DirectDraw object.

dwFlags
Not currently used.

Return Values

DDERR_ALREADYINITIALIZED

See Also

AddRefAddRef, **CreateClipper**, **QueryInterfaceQueryInterfaceQueryInterface**, **Release**

IsClipListChanged

If an hWnd is associated with a DirectDrawClipper, this member can be used to monitor the status of the clip list.

```
HRESULT IsClipListChanged(
    LPDIRECTDRAWCLIPPER lpDDClipper,
    BOOL FAR *lpbChanged )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper.

lpbChanged
A pointer to a Boolean value set equal to TRUE if the clip list has changed.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

QueryInterface

This member is part of the **IUnknown** interface inherited by the DirectDrawClipper. It is used to increase the reference count of the DirectDrawClipper object. This is the member that applications use to determine whether the DirectDrawClipper object supports additional interfaces that they may be interested in. An application can ask the DirectDrawClipper object if it supports a particular COM interface, and if it does, the application may begin using that interface immediately. If the application does not want to use that interface it must call Release to free it. This member allows DirectDrawClipper objects to be extended by Microsoft and third parties without breaking, or interfering with, each other's existing or future functionality.

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

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure which was returned to the application when the DirectDrawClipper was created using the CreateClipper member of the DirectDraw object.

riid
Points to the GUID used as the interface identifier.

ppvObj
Points to a pointer which will receive the interface pointer if the request is successful.

Return Values

DD_OK **DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS

See Also

AddRef, **Release**, **Initialize**

Release

This member is part of the **IUnknown** interface inherited by the DirectDrawClipper. It is used to decrease the reference count of the DirectDrawClipper object. When the DirectDrawClipper object is initially created, its reference count is set to one. Each time **Release** is called by an application the DirectDrawClipper object reduces the reference count by one. The DirectDrawClipper object deallocates itself when its reference count goes to zero. The AddRef member is used to increase the reference count every time a new application binds to the DirectDrawClipper object.

```
DWORD Release(
    LPDIRECTDRAWCLIPPER lpDDClipper )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure which was returned to the client when the DirectDraw member **CreateClipper** was called to create the DirectDrawClipper.

Return Values

SUCCESS	**Reference count of the object.**
FAILURE	**Zero**

See Also

AddRef, **Initialize**, **QueryInterface**

SetClipList

Set or delete the clip list used by the Blt, BltBatch, and UpdateOverlay members of surfaces to which the parent DirectDrawClipper is attached. The clip list is a series of rectangles that describe the areas of the surface that are visible. The clip list cannot be set if there is already an hWnd associated with the DirectDrawClipper object. Note that the BltFast member cannot clip.

```
HRESULT SetClipList(
    LPDIRECTDRAWCLIPPER lpDDClipper,
    LPRGNDATA lpClipList,
    DWORD dwFlags )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper.

lpClipList
This is either a pointer to a valid **RGNDATA** or NULL. If it is NULL, and there is an existing clip list associated with the DirectDrawClipper, it will be deleted.

dwFlags
Not used at this time.

Return Values

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

See Also

GetClipList, Blt, BltFast, BltBatch, UpdateOverlay

SetHWnd

Set the HWnd that will be used to obtain clipping information.

```
HRESULT SetHWnd(
    LPDIRECTDRAWCLIPPER lpDDClipper,
    DWORD dwFlags,
    HWND hWnd )
```

Parameters

lpDDClipper
Points to the **DIRECTDRAWCLIPPER** structure representing the DirectDrawClipper.

dwFlags
Not in use.

hWnd
The hWnd that represents the clipping information.

Return Values

DD_OK	**DDERR_INVALIDOBJECT**
DDERR_INVALIDPARAMS	**DDERR_OUTOFMEMORY**

See Also

GetHWnd

Return Codes

DD_OK
DDERR_ALREADYINITIALIZED
DDERR_BLTFASTCANTCLIP
DDERR_CANNOTATTACHSURFACE
DDERR_CANNOTDETACHSURFACE
DDERR_CANTCREATEDC
DDERR_CANTDUPLICATE
DDERR_CLIPPERISUSINGHWND
DDERR_COLORKEYNOTSET
DDERR_CURRENTLYNOTAVAIL
DDERR_DIRECTDRAWALREADYCREATED
DDERR_EXCEPTION
DDERR_EXCLUSIVEMODEALREADYSET
DDERR_GENERIC
DDERR_HEIGHTALIGN
DDERR_HWNDALREADYSET
DDERR_HWNDSUBCLASSED
DDERR_IMPLICITLYCREATED
DDERR_INCOMPATIBLEPRIMARY
DDERR_INVALIDCAPS
DDERR_INVALIDCLIPLIST
DDERR_INVALIDDIRECTDRAWGUID
DDERR_INVALIDMODE
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDPIXELFORMAT
DDERR_INVALIDPOSITION
DDERR_INVALIDRECT
DDERR_LOCKEDSURFACES
DDERR_NO3D
DDERR_NOALPHAHW
DDERR_NOANTITEARHW
DDERR_NOBLTHW
DDERR_NOBLTQUEUEHW
DDERR_NOCLIPLIST
DDERR_NOCLIPPERATTACHED

DDERR_NOCOLORCONVHW
DDERR_NOCOLORKEY
DDERR_NOCOLORKEYHW
DDERR_NOCOOPERATIVELEVELSET
DDERR_NODC
DDERR_NODDROPSHW
DDERR_NODIRECTDRAWHW
DDERR_NOEMULATION
DDERR_NOEXCLUSIVEMODE
DDERR_NOFLIPHW
DDERR_NOGDI
DDERR_NOHWND
DDERR_NOMIRRORHW
DDERR_NOOVERLAYDEST
DDERR_NOOVERLAYHW
DDERR_NOPALETTEATTACHED
DDERR_NOPALETTEHW
DDERR_NORASTEROPHW
DDERR_NOROTATIONHW
DDERR_NOSTRETCHHW
DDERR_NOT4BITCOLOR
DDERR_NOT4BITCOLORINDEX
DDERR_NOT8BITCOLOR
DDERR_NOTAOVERLAYSURFACE
DDERR_NOTEXTUREHW
DDERR_NOTFLIPPABLE
DDERR_NOTFOUND
DDERR_NOTLOCKED
DDERR_NOTPALETTIZED
DDERR_NOVSYNCHW
DDERR_NOZBUFFERHW
DDERR_NOZOVERLAYHW
DDERR_OUTOFCAPS
DDERR_OUTOFMEMORY
DDERR_OUTOFVIDEOMEMORY
DDERR_OVERLAYCANTCLIP
DDERR_OVERLAYCOLORKEYONLYONEACTIVE

DDERR_OVERLAYNOTVISIBLE
DDERR_PALETTEBUSY
DDERR_PRIMARYSURFACEALREADYEXISTS
DDERR_REGIONTOOSMALL
DDERR_SURFACEALREADYATTACHED
DDERR_SURFACEALREADYDEPENDENT
DDERR_SURFACEBUSY
DDERR_SURFACEISOBSCURED
DDERR_SURFACELOST
DDERR_SURFACENOTATTACHED
DDERR_TOOBIGHEIGHT
DDERR_TOOBIGSIZE
DDERR_TOOBIGWIDTH
DDERR_UNSUPPORTED
DDERR_UNSUPPORTEDFORMAT
DDERR_UNSUPPORTEDMASK
DDERR_VERTICALBLANKINPROGRESS
DDERR_WASSTILLDRAWING
DDERR_WRONGMODE
DDERR_XALIGN

DD_OK

Status OK, request completed successfully.

DDERR_ALREADYINITIALIZED

This object is already initialized.

DDERR_BLTFASTCANTCLIP

Return if a clipper object is attached to the source surface passed into a BltFast call.

DDERR_CANNOTATTACHSURFACE

This surface can not be attached to the requested surface.

DDERR_CANNOTDETACHSURFACE

This surface can not be detached from the requested surface.

DDERR_CANTCREATEDC

Windows can not create any more DCs

DDERR_CANTDUPLICATE

Can't duplicate primary & 3D surfaces, or surfaces that are implicitly created.

DDERR_CLIPPERISUSINGHWND

An attempt was made to set a cliplist for a clipper object that is already monitoring an hwnd.

DDERR_COLORKEYNOTSET

No src color key specified for this operation.

DDERR_CURRENTLYNOTAVAIL

Support is currently not available.

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

Generic failure.

DDERR_HEIGHTALIGN

Height of rectangle provided is not a multiple of reqd alignment.

DDERR_HWNDALREADYSET

The CooperativeLevel HWND has already been set. It can not be reset while the process has surfaces or palettes created.

DDERR_HWNDSUBCLASSED

HWND used by DirectDraw CooperativeLevel has been subclassed, this prevents DirectDraw from restoring state.

DDERR_IMPLICITLYCREATED

This surface can not be restored because it is an implicitly created surface.

DDERR_INCOMPATIBLEPRIMARY

Unable to match primary surface creation request with existing primary surface.

DDERR_INVALIDCAPS

One or more of the caps bits passed to the callback are incorrect.

DDERR_INVALIDCLIPLIST

DirectDraw does not support the provided cliplist.

DDERR_INVALIDDIRECTDRAWGUID

The GUID passed to DirectDrawCreate 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 function are incorrect.

DDERR_INVALIDPIXELFORMAT

The pixel format was invalid as specified.

DDERR_INVALIDPOSITION

Returned when the position of the overlay on the destination is no longer legal for that destination.

DDERR_INVALIDRECT

Rectangle provided was invalid.

DDERR_LOCKEDSURFACES

Operation could not be carried out because one or more surfaces are locked.

DDERR_NO3D

There is no 3D present.

DDERR_NOALPHAHW

Operation could not be carried out because there is no alpha accleration hardware present or available.

DDERR_NOANTITEARHW

Operation could not be carried out because there is no hardware support for synchronizing blits to avoid tearing.

DDERR_NOBLTHW

No blitter hardware present.

DDERR_NOBLTQUEUEHW

Operation could not be carried out because there is no hardware support for asynchronous blitting.

DDERR_NOCLIPLIST

No cliplist available.

DDERR_NOCLIPPERATTACHED

No clipper object attached to surface object.

DDERR_NOCOLORCONVHW

Operation could not be carried out because there is no color conversion hardware present or available.

DDERR_NOCOLORKEY

Surface doesn't currently have a color key

DDERR_NOCOLORKEYHW

Operation could not be carried out because there is no hardware support of the destination color key.

DDERR_NOCOOPERATIVELEVELSET

Create function called without DirectDraw object method SetCooperativeLevel being called.

DDERR_NODC

No DC was ever created for this surface.

DDERR_NODDROPSHW

No DirectDraw ROP hardware.

DDERR_NODIRECTDRAWHW

A hardware-only DirectDraw object creation was attempted but the driver did not support any hardware.

DDERR_NOEMULATION

Software emulation not available.

DDERR_NOEXCLUSIVEMODE

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

There is no GDI present.

DDERR_NOHWND

Clipper notification requires an HWND or no HWND has previously been set as the CooperativeLevel HWND.

DDERR_NOMIRRORHW

Operation could not be carried out because there is no hardware present or available.

DDERR_NOOVERLAYDEST

Returned when GetOverlayPosition is called on an overlay that UpdateOverlay has never been called on to establish a destination.

DDERR_NOOVERLAYHW

Operation could not be carried out because there is no overlay hardware present or available.

DDERR_NOPALETTEATTACHED

No palette object attached to this surface.

DDERR_NOPALETTEHW

No hardware support for 16 or 256 color palettes.

DDERR_NORASTEROPHW

Operation could not be carried out because there is no appropriate raster op hardware present or available.

DDERR_NOROTATIONHW

Operation could not be carried out because there is no rotation hardware present or available.

DDERR_NOSTRETCHHW

Operation could not be carried out because there is no hardware support for stretching.

DDERR_NOT4BITCOLOR

DirectDrawSurface is not in 4 bit color palette and the requested operation requires 4 bit color palette.

DDERR_NOT4BITCOLORINDEX

DirectDrawSurface is not in 4 bit color index palette and the requested operation requires 4 bit color index palette.

DDERR_NOT8BITCOLOR

DirectDrawSurface is not in 8 bit color mode and the requested operation requires 8 bit color.

DDERR_NOTAOVERLAYSURFACE

Returned when an overlay member is called for a non-overlay surface.

DDERR_NOTEXTUREHW

Operation could not be carried out because there is no texture mapping hardware present or available.

DDERR_NOTFLIPPABLE

An attempt has been made to flip a surface that is not flippable.

DDERR_NOTFOUND

Requested item was not found.

DDERR_NOTLOCKED

Surface was not locked. An attempt to unlock a surface that was not locked at all, or by this process, has been attempted.

DDERR_NOTPALETTIZED

The surface being used is not a palette-based surface.

DDERR_NOVSYNCHW

Operation could not be carried out because there is no hardware support for vertical blank synchronized operations.

DDERR_NOZBUFFERHW

Operation could not be carried out because there is no hardware support for zbuffer blitting.

DDERR_NOZOVERLAYHW

Overlay surfaces could not 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 memory to perform the operation.

DDERR_OVERLAYCANTCLIP

The hardware does not support clipped overlays.

DDERR_OVERLAYCOLORKEYONLYONEACTIVE

Can only have ony color key active at one time for overlays.

DDERR_OVERLAYNOTVISIBLE

Returned when GetOverlayPosition is called on a hidden overlay.

DDERR_PALETTEBUSY

Access to this palette is being refused because the palette is already locked by another thread.

DDERR_PRIMARYSURFACEALREADYEXISTS

This process already has created a primary surface.

DDERR_REGIONTOOSMALL

Region passed to Clipper::GetClipList is too small.

DDERR_SURFACEALREADYATTACHED

This surface is already attached to the surface it is being attached to.

DDERR_SURFACEALREADYDEPENDENT

This surface is already a dependency of the surface it is being made a dependency of.

DDERR_SURFACEBUSY

Access to this surface is being refused because the surface is already locked by another thread.

DDERR_SURFACEISOBSCURED

Access to surface refused because the surface is obscured.

DDERR_SURFACELOST

Access to this surface is being refused because the surface memory is gone. The DirectDrawSurface object representing this surface should have **Restore** called on it.

DDERR_SURFACENOTATTACHED

The requested surface is not attached.

DDERR_TOOBIGHEIGHT

Height requested by DirectDraw is too large.

DDERR_TOOBIGSIZE

Size requested by DirectDraw is too large -- the individual height and width are OK.

DDERR_TOOBIGWIDTH

Width requested by DirectDraw is too large.

DDERR_UNSUPPORTED

Action not supported.

DDERR_UNSUPPORTEDFORMAT

FOURCC format requested is unsupported by DirectDraw.

DDERR_UNSUPPORTEDMASK

Bitmask in the pixel format requested is unsupported by DirectDraw.

DDERR_VERTICALBLANKINPROGRESS

Vertical blank is in progress.

DDERR_WASSTILLDRAWING

Informs DirectDraw that the previous Blt which is transfering information to or from this Surface is incomplete.

DDERR_WRONGMODE

This surface can not be restored because it was created in a different mode.

DDERR_XALIGN

Rectangle provided was not horizontally aligned on required boundary.