DirectDraw Reference

This section contains reference information for the API elements that DirectDraw provides. Reference material is divided into the following categories:

- Interfaces
- Functions
- Callback Functions
- Structures
- Return Values
- Pixel Format Masks
- Four Character Codes (FOURCC)

Interfaces

This section contains reference information about the interfaces used with the DirectDraw component. The following interfaces are covered:

- **IDDVideoPortContainer**
- **IDirectDraw2**
- **IDirectDrawClipper**
- **IDirectDrawColorControl**
- **IDirectDrawPalette**
- **IDirectDrawSurface3**
- **IDirectDrawVideoPort**

IDDVideoPortContainer

Applications use the methods of the **IDDVideoPortContainer** interface to create and manipulate DirectDrawVideoPort objects.

The methods of the **IDDVideoPortContainer** interface can be organized into the following groups:

Creating objects	**CreateVideoPort**
Video ports	**EnumVideoPorts** **QueryVideoPortStatus**
Connections	**GetVideoPortConnectInfo**

The **IDDVideoPortContainer** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef

QueryInterface

Release

You can use the LPDDVIDEOPORTCONTAINER data type to declare a variable that contains a pointer to an **IDDVideoPortContainer** interface. The Dvp.h header file declares the LPDDVIDEOPORTCONTAINER with the following code:

```
typedef struct IDDVideoPortContainer    FAR *LPDDVIDEOPORTCONTAINER;
```

IDDVideoPortContainer::CreateVideoPort

The **IDDVideoPortContainer::CreateVideoPort** method creates a DirectDrawVideoPort object.

```
HRESULT CreateVideoPort(
  DWORD dwFlags,
  LPDDVIDEOPORTDESC lpDDVideoPortDesc,
  LPDIRECTDRAWVIDEOPORT FAR *lplpDDVideoPort,
  IUnknown FAR *pUnkOuter
);
```

Parameters

dwFlags
Reserved for future use. This parameter must be zero.

lpDDVideoPortDesc
Address of a **DDVIDEOPORTDESC** structure that describes the VideoPort object to be created.

lplpDDVideoPort
Address of a variable that will be filled with a pointer to the new DirectDrawVideoPort object's **IDirectDrawVideoPort** interface if the call succeeds.

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS
DDERR_NOCOOPERATIVELEVELSET
DDERR_OUTOFCAPS
DDERR_OUTOFMEMORY

IDDVideoPortContainer::EnumVideoPorts

The **IDDVideoPortContainer::EnumVideoPorts** method enumerates all of the video ports that the hardware exposes that are compatible with a provided video port description.

```
HRESULT EnumVideoPorts(
  DWORD dwFlags,
  LPDDVIDEOPORTCAPS lpDDVideoPortCaps,
  LPVOID lpContext,
  LPENUMVIDEOCALLBACK lpEnumVideoCallback
);
```

Parameters

dwFlags
Reserved for future use. This parameter must be zero.

lpDDVideoPortCaps
Pointer to a **DDVIDEOPORTCAPS** structure that will be checked against the available video ports. If this parameter is NULL, all video ports will be enumerated.

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

lpEnumVideoCallback
Address of the **EnumVideoCallback** function that will be called each time a match is found.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

IDDVideoPortContainer::GetVideoPortConnectInfo

The **IDDVideoPortContainer::GetVideoPortConnectInfo** method retrieves the connection information supported by all video ports.

```
HRESULT GetVideoPortConnectInfo(
  DWORD dwPortId,
  LPDWORD lpNumEntries,
  LPDDVIDEOPORTCONNECT lpConnectInfo
);
```

Parameters

dwPortId
Video port ID of the video port for which the connection information will be retrieved.

lpNumEntries
Address of a variable containing the number of entries that the array at *lpConnectInfo* can hold. If this number is less than the total number of connections, the method fills the array with as many entries as will fit, sets the value at *lpNumEntries* to indicate the total number of connections, and returns DDERR_MOREDATA.

lpConnectInfo
Address of an array of **DDVIDEOPORTCONNECT** structures that will be filled with the connection options supported by the specified video port. If this parameter is NULL, the method sets *lpNumEntries* to indicate the total number of connections that the video port supports, then returns DD_OK.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_MOREDATA

IDDVideoPortContainer::QueryVideoPortStatus

The **IDDVideoPortContainer::QueryVideoPortStatus** method is not currently implemented.

```
HRESULT QueryVideoPortStatus(
  DWORD dwPortId,
  LPDDVIDEOPORTSTATUS lpVPStatus
);
```

Parameters

dwPortId
Video port ID of the video port for which the status information will be retrieved.

lpVPStatus
Address of a **DDVIDEOPORTSTATUS** structure that will be filled with information about the status of the specified video port.

Return Values

This method returns DDERR_UNSUPPORTED.

IDirectDraw2

Applications use the methods of the **IDirectDraw2** interface to create DirectDraw objects and work with system-level variables. This section is a reference to the methods of this interface. For a conceptual overview, see The DirectDraw Object.

The methods of the **IDirectDraw2** interface can be organized into the following groups:

Allocating memory	**Compact**
	Initialize
Creating objects	**CreateClipper**
	CreatePalette
	CreateSurface
Device capabilities	**GetCaps**
Display modes	**EnumDisplayModes**
	GetDisplayMode
	GetMonitorFrequency
	RestoreDisplayMode
	SetDisplayMode
	WaitForVerticalBlank
Display status	**GetScanLine**

	GetVerticalBlankStatus
Miscellaneous	**GetAvailableVidMem**
	GetFourCCCodes
Setting behavior	**SetCooperativeLevel**
Surfaces	**DuplicateSurface**
	EnumSurfaces
	FlipToGDISurface
	GetGDISurface

The **IDirectDraw2** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

You can use the LPDIRECTDRAW or LPDIRECTDRAW data types to declare a variable that contains a pointer to an **IDirectDraw** or **IDirectDraw2** interface. The Ddraw.h header file declares these data types with the following code:

```
typedef struct IDirectDraw     FAR *LPDIRECTDRAW;
typedef struct IDirectDraw2    FAR *LPDIRECTDRAW2;
```

IDirectDraw2::Compact

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

```
HRESULT Compact();
```

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOEXCLUSIVEMODE
DDERR_SURFACEBUSY

Remarks

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

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

IDirectDraw2::CreateClipper

The **IDirectDraw2::CreateClipper** method creates a DirectDrawClipper object.

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

Parameters

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

lplpDDClipper
Address of a pointer that will be filled with the address of the new DirectDrawClipper object if this method returns successfully.

pUnkOuter
Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw2::CreateClipper** returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_NOCOOPERATIVELEVELSET

DDERR_OUTOFMEMORY

Remarks

The DirectDrawClipper object can be attached to a DirectDrawSurface and used during **IDirectDrawSurface3::Blt**, **IDirectDrawSurface3::BltBatch**, and **IDirectDrawSurface3::UpdateOverlay** operations.

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

See Also

IDirectDrawSurface3::GetClipper, **IDirectDrawSurface3::SetClipper**

IDirectDraw2::CreatePalette

The **IDirectDraw2::CreatePalette** method creates a DirectDrawPalette object for this DirectDraw object.

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

Parameters

dwFlags

One or more of the following flags:

DDPCAPS_1BIT	Indicates that the index is 1 bit. There are two entries in the color table.
DDPCAPS_2BIT	Indicates that the index is 2 bits. There are four entries in the color table.
DDPCAPS_4BIT	Indicates that the index is 4 bits. There are 16 entries in the color table.
DDPCAPS_8BITENTRIES	Indicates that the index refers to an 8-bit color index. This flag is valid only when used with the DDPCAPS_1BIT, DDPCAPS_2BIT, or DDPCAPS_4BIT flag, and when the target surface is in 8 bpp. Each color entry is 1 byte long and is an index to a destination surface's 8-bpp palette.
DDPCAPS_8BIT	Indicates that the index is 8 bits. There are 256 entries in the color table.
DDPCAPS_ALLOW256	Indicates that this palette can have all 256 entries defined.

lpColorTable

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

lplpDDPalette

Address of a pointer that will be filled with the address of the new DirectDrawPalette object if this method returns successfully.

pUnkOuter

Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw2::CreatePalette** returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOCOOPERATIVELEVELSET
DDERR_OUTOFMEMORY
DDERR_UNSUPPORTED

IDirectDraw2::CreateSurface

The **IDirectDraw2::CreateSurface** method creates a DirectDrawSurface object for this DirectDraw object.

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

Parameters

lpDDSurfaceDesc
Address of the **DDSURFACEDESC** structure that describes the requested surface. You should set any unused members of **DDSURFACEDESC** to zero before calling this method. A **DDSCAPS** structure is a member of **DDSURFACEDESC**.

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

pUnkOuter
Allows for future compatibility with COM aggregation features. Presently, however, **IDirectDraw2::CreateSurface** returns an error if this parameter is anything but NULL.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INCOMPATIBLEPRIMARY
DDERR_INVALIDCAPS
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDPIXELFORMAT
DDERR_NOALPHAHW
DDERR_NOCOOPERATIVELEVELSET
DDERR_NODIRECTDRAWHW
DDERR_NOEMULATION
DDERR_NOEXCLUSIVEMODE
DDERR_NOFLIPHW
DDERR_NOMIPMAPHW
DDERR_NOOVERLAYHW
DDERR_NOZBUFFERHW
DDERR_OUTOFMEMORY
DDERR_OUTOFVIDEOMEMORY
DDERR_PRIMARYSURFACEALREADYEXISTS
DDERR_UNSUPPORTEDMODE

IDirectDraw2::DuplicateSurface

The **IDirectDraw2::DuplicateSurface** method duplicates a DirectDrawSurface object.

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

Parameters

lpDDSurface
Address of the DirectDrawSurface structure to be duplicated.

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CANTDUPLICATE
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_OUTOFMEMORY
DDERR_SURFACELOST

Remarks

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

IDirectDraw2::EnumDisplayModes

The **IDirectDraw2::EnumDisplayModes** method enumerates all of the display modes the hardware exposes through the DirectDraw object that are compatible with a provided surface description.

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

Parameters

dwFlags

DDEDM_REFRESHRATES	Enumerates modes with different refresh rates. **IDirectDraw2::EnumDisplayModes** guarantees that a partic mode will be enumerated only once. This flag specifies wheth refresh rate is taken into account when determining if a mode unique.
DDEDM_STANDARDVGAMODES	Enumerates Mode 13 in addition to the 320x200x8 Mode X m

lpDDSurfaceDesc

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

lpContext

Address of an application-defined structure that will be passed to each enumeration member.

lpEnumModesCallback

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

Remarks

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

See Also

IDirectDraw2::GetDisplayMode, **IDirectDraw2::SetDisplayMode**, **IDirectDraw2::RestoreDisplayMode**

IDirectDraw2::EnumSurfaces

The **IDirectDraw2::EnumSurfaces** method enumerates all of the existing or possible surfaces that meet the search criterion specified.

```
HRESULT EnumSurfaces(
  DWORD dwFlags,
  LPDDSURFACEDESC lpDDSD,
  LPVOID lpContext,
  LPDDENUMSURFACESCALLBACK lpEnumSurfacesCallback
);
```

Parameters

dwFlags

One of the following flags:

DDENUMSURFACES_ALL

Enumerates all of the surfaces that meet the search criterion.

DDENUMSURFACES_CANBECREATED

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

DDENUMSURFACES_DOESEXIST

Enumerates the already existing surfaces that meet the search criterion.

DDENUMSURFACES_MATCH

Searches for any surface that matches the surface description.

DDENUMSURFACES_NOMATCH

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

lpDDSD

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

lpContext

Address of an application-defined structure that will be passed to each enumeration member.

lpEnumSurfacesCallback

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

Remarks

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

As part of the **IDirectDraw** interface, this method did not support any values other than zero for the *dwFlags* parameter.

IDirectDraw2::FlipToGDISurface

The **IDirectDraw2::FlipToGDISurface** method makes the surface that GDI writes to the primary surface.

```
HRESULT FlipToGDISurface();
```

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOTFOUND

Remarks

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

See Also

IDirectDraw2::GetGDISurface

IDirectDraw2::GetAvailableVidMem

The **IDirectDraw2::GetAvailableVidMem** method retrieves the total amount of display memory available and the amount of display memory currently free for a given type of surface.

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

Parameters

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

lpdwTotal
Address of a variable that will be filled with the total amount of display memory available.

lpdwFree
Address of a variable that will be filled with the amount of display memory currently free that can be allocated for a surface that matches the capabilities specified by the structure at *lpDDSCaps*.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDCAPS
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NODIRECTDRAWHW

If NULL is passed to either *lpdwTotal* or *lpdwFree*, the value for that parameter is not returned.

Remarks

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

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

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

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

determining how much memory is available for dynamic use (such as texture mapping).

This method was not implemented in the **IDirectDraw** interface.

IDirectDraw2::GetCaps

The **IDirectDraw2::GetCaps** method fills in the capabilities of the device driver for the hardware and the hardware-emulation layer (HEL).

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

Parameters

lpDDDriverCaps
Address of a **DDCAPS** structure that will be filled with the capabilities of the hardware, as reported by the device driver. Set this parameter to NULL if device driver capabilities are not to be retrieved.

lpDDHELCaps
Address of a **DDCAPS** structure that will be filled with the capabilities of the HEL. Set this parameter to NULL if HEL capabilities are not to be retrieved.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

You can only set one of the two parameters to NULL to exclude it. If you set both to NULL the method will fail, returning DDERR_INVALIDPARAMS.

IDirectDraw2::GetDisplayMode

The **IDirectDraw2::GetDisplayMode** method retrieves the current display mode.

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

Parameters

lpDDSurfaceDesc
Address of a **DDSURFACEDESC** structure that will be filled with a description of the surface.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTEDMODE

Remarks

An application should not save the information returned by this method to restore the display mode on clean-up. The application should use the **IDirectDraw2::RestoreDisplayMode** method to restore the mode on clean-up, thereby avoiding mode-setting conflicts that could arise in a multiprocess environment.

See Also

IDirectDraw2::SetDisplayMode, **IDirectDraw2::RestoreDisplayMode**, **IDirectDraw2::EnumDisplayModes**

IDirectDraw2::GetFourCCCodes

The **IDirectDraw2::GetFourCCCodes** method retrieves the FOURCC codes supported by the DirectDraw object. This method can also retrieve the number of codes supported.

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

Parameters

lpNumCodes
Address of a variable that contains the number of entries that the array pointed to by *lpCodes* can hold. If the number of entries is too small to accommodate all the codes, *lpNumCodes* is set to the required number and the array pointed to by *lpCodes* is filled with all that fits.

lpCodes
Address of an array of variables that will be filled with FOURCC codes supported by this DirectDraw object. If you specify NULL, *lpNumCodes* is set to the number of supported FOURCC codes and the method will return.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

IDirectDraw2::GetGDISurface

The **IDirectDraw2::GetGDISurface** method retrieves the DirectDrawSurface object that currently represents the surface memory that GDI is treating as the primary surface.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOTFOUND

See Also

IDirectDraw2::FlipToGDISurface

IDirectDraw2::GetMonitorFrequency

The **IDirectDraw2::GetMonitorFrequency** method retrieves the frequency of the monitor being driven by the DirectDraw object.

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

Parameters

lpdwFrequency
Address of the variable that will be filled with the monitor frequency, reported in Hz.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

IDirectDraw2::GetScanLine

The **IDirectDraw2::GetScanLine** method retrieves the scan line that is currently being drawn on the monitor.

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

Parameters

lpdwScanLine
Address of the variable that will contain the scan line the display is currently on.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

DDERR_UNSUPPORTED
DDERR_VERTICALBLANKINPROGRESS

Remarks

Scan lines are reported as zero-based integers. The returned scan line value is between 0 and *n*, where scan line 0 is the first visible scan line on the screen and *n* is the last visible scan line, plus any scan lines that occur during vblank. So, in a case where an application is running at 640×480, and there are 12 scan lines during vblank, the values returned by this method will range from 0 to 491.

See Also

IDirectDraw2::GetVerticalBlankStatus, **IDirectDraw2::WaitForVerticalBlank**

IDirectDraw2::GetVerticalBlankStatus

The **IDirectDraw2::GetVerticalBlankStatus** method retrieves the status of the vertical blank.

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

Parameters

lpbIsInVB
Address of the variable that will be filled with the status of the vertical blank. This parameter is TRUE if a vertical blank is occurring, and FALSE otherwise.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

To synchronize with the vertical blank, use the **IDirectDraw2::WaitForVerticalBlank** method.

See Also

IDirectDraw2::GetScanLine, **IDirectDraw2::WaitForVerticalBlank**

IDirectDraw2::Initialize

The **IDirectDraw2::Initialize** method initializes the DirectDraw object that was created by using the **CoCreateInstance** COM function.

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

Parameters

lpGUID
Address of the globally unique identifier (GUID) used as the interface identifier.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_ALREADYINITIALIZED
DDERR_DIRECTDRAWALREADYCREATED
DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NODIRECTDRAWHW
DDERR_NODIRECTDRAWSUPPORT
DDERR_OUTOFMEMORY

This method is provided for compliance with the Component Object Model (COM) protocol. If the **DirectDrawCreate** function was used to create the DirectDraw object, this method returns DDERR_ALREADYINITIALIZED. If **IDirectDraw2::Initialize** is not called when using **CoCreateInstance** to create the DirectDraw object, any method that is called afterward returns DDERR_NOTINITIALIZED.

Remarks

For more information about using **IDirectDraw2::Initialize** with **CoCreateInstance**, see Creating DirectDraw Objects by Using CoCreateInstance.

IDirectDraw2::RestoreDisplayMode

The **IDirectDraw2::RestoreDisplayMode** method resets the mode of the display device hardware for the primary surface to what it was before the

IDirectDraw2::SetDisplayMode method was called. Exclusive-level access is required to use this method.

HRESULT RestoreDisplayMode();

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_LOCKEDSURFACES
DDERR_NOEXCLUSIVEMODE

See Also

IDirectDraw2::SetDisplayMode, **IDirectDraw2::EnumDisplayModes**, **IDirectDraw2::SetCooperativeLevel**

IDirectDraw2::SetCooperativeLevel

The **IDirectDraw2::SetCooperativeLevel** method determines the top-level behavior of the application.

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

Parameters

hWnd
Window handle used for the application. This parameter can be NULL when the DDSCL_NORMAL flag is specified in the *dwFlags* parameter.

dwFlags
One or more of the following flags:

DDSCL_ALLOWMODEX
Allows the use of Mode X display modes. This flag must be used with the DDSCL_EXCLUSIVE and DDSCL_FULLSCREEN flags.

DDSCL_ALLOWREBOOT
Allows CTRL+ALT+DEL to function while in exclusive (full-screen) mode.

DDSCL_EXCLUSIVE

Requests the exclusive level. This flag must be used with the DDSCL_FULLSCREEN flag.

DDSCL_FULLSCREEN

Indicates that the exclusive-mode owner will be responsible for the entire primary surface. GDI can be ignored. This flag must be used with the DDSCL_EXCLUSIVE flag.

DDSCL_NORMAL

Indicates that the application will function as a regular Windows application. This flag cannot be used with the DDSCL_ALLOWMODEX, DDSCL_EXCLUSIVE, or DDSCL_FULLSCREEN flags.

DDSCL_NOWINDOWCHANGES

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_EXCLUSIVEMODEALREADYSET

DDERR_HWNDALREADYSET

DDERR_HWNDSUBCLASSED

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_OUTOFMEMORY

Remarks

An application must set either the DDSCL_EXCLUSIVE or DDSCL_NORMAL flag.

The DDSCL_EXCLUSIVE flag must be set to call functions that can have drastic performance consequences for other applications. For more information, see Cooperative Levels.

Interaction between this method and the **IDirectDraw2::SetDisplayMode** method differs from their **IDirectDraw** counterparts. For more information, see Cooperative Levels and Display Modes with IDirectDraw2.

See Also

IDirectDraw2::SetDisplayMode, **IDirectDraw2::Compact**, **IDirectDraw2::EnumDisplayModes**, Mode X and Mode 13 Display Modes.

IDirectDraw2::SetDisplayMode

The **IDirectDraw2::SetDisplayMode** method sets the mode of the display-device hardware.

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

Parameters

dwWidth and *dwHeight*
Width and height of the new mode.

dwBPP
Bits per pixel (bpp) of the new mode.

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

dwFlags
Flags describing additional options. Currently, the only valid flag is DDSDM_STANDARDVGAMODE, which causes the method to set Mode 13 instead of Mode X 320x200x8 mode. If you are setting another resolution, bit depth, or a Mode X mode, do not use this flag and set the parameter to 0.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDMODE
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_LOCKEDSURFACES
DDERR_NOEXCLUSIVEMODE
DDERR_SURFACEBUSY
DDERR_UNSUPPORTED
DDERR_UNSUPPORTEDMODE
DDERR_WASSTILLDRAWING

Remarks

If another application changes the display mode, the primary surface will be lost and will return DDERR_SURFACELOST until it is recreated to match the new display mode.

As part of the **IDirectDraw** interface, this method did not include the *dwRefreshRate* and *dwFlags* parameters.

See Also

IDirectDraw2::RestoreDisplayMode, **IDirectDraw2::GetDisplayMode**, **IDirectDraw2::EnumDisplayModes**, **IDirectDraw2::SetCooperativeLevel**, Cooperative Levels and Display Modes with IDirectDraw2, Setting Display Modes

IDirectDraw2::WaitForVerticalBlank

The **IDirectDraw2::WaitForVerticalBlank** method helps the application synchronize itself with the vertical-blank interval.

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

Parameters

dwFlags

Determines how long to wait for the vertical blank.

DDWAITVB_BLOCKBEGIN

Returns when the vertical-blank interval begins.

DDWAITVB_BLOCKBEGINEVENT

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

DDWAITVB_BLOCKEND

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

hEvent

Handle of the event to be triggered when the vertical blank begins. This parameter is not currently used.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED
DDERR_WASSTILLDRAWING

See Also

IDirectDraw2::GetVerticalBlankStatus, **IDirectDraw2::GetScanLine**

IDirectDrawClipper

Applications use the methods of the **IDirectDrawClipper** interface to manage clip lists. This section is a reference to the methods of this interface. For a conceptual overview, see Clippers.

The methods of the **IDirectDrawClipper** interface can be organized into the following groups:

Allocating memory	**Initialize**
Clip list	**GetClipList**
	IsClipListChanged
	SetClipList
	SetHWnd
Handles	**GetHWnd**

The **IDirectDrawClipper** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

You can use the LPDIRECTDRAWCLIPPER data type to declare a variable that contains a pointer to an **IDirectDrawClipper** interface. The Ddraw.h header file declares these data types with the following code:

```
typedef struct IDirectDrawClipper    FAR *LPDIRECTDRAWCLIPPER;
```

IDirectDrawClipper::GetClipList

The **IDirectDrawClipper::GetClipList** method retrieves a copy of the clip list associated with a DirectDrawClipper object. A subset of the clip list can be selected by passing a rectangle that clips the clip list.

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

Parameters

lpRect
Address of a rectangle that will be used to clip the clip list. This parameter can be NULL to retrieve the entire clip list.

lpClipList
Address of an **RGNDATA** structure that will contain the resulting copy of the clip list. If this parameter is NULL, the method fills the variable at *lpdwSize* to the number of bytes necessary to hold the entire clip list.

lpdwSize
Size of the resulting clip list. When retrieving the clip list, this parameter is the size of the buffer at *lpClipList*. When *lpClipList* is NULL, the variable at *lpdwSize* receives the required size of the buffer, in bytes.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDCLIPLIST
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOCLIPLIST
DDERR_REGIONTOOSMALL

Remarks

The **RGNDATA** structure used with this method has the following syntax.

```
typedef struct _RGNDATA {
    RGNDATAHEADER rdh;
    char          Buffer[1];
```

```
} RGNDATA;
```

The **rdh** member of the **RGNDATA** structure is an **RGNDATAHEADER** structure that has the following syntax.

```
typedef struct _RGNDATAHEADER {
    DWORD dwSize;
    DWORD iType;
    DWORD nCount;
    DWORD nRgnSize;
    RECT  rcBound;
} RGNDATAHEADER;
```

For more information about these structures, see the documentation in the Platform SDK.

See Also

IDirectDrawClipper::SetClipList

IDirectDrawClipper::GetHWnd

The **IDirectDrawClipper::GetHWnd** method retrieves the window handle previously associated with this DirectDrawClipper object by the **IDirectDrawClipper::SetHWnd** method.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirectDrawClipper::SetHWnd

IDirectDrawClipper::Initialize

The **IDirectDrawClipper::Initialize** method initializes a DirectDrawClipper object that was created by using the **CoCreateInstance** COM function.

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

Parameters

lpDD
Address of the DirectDraw structure that represents the DirectDraw object. If this parameter is set to NULL, an independent DirectDrawClipper object is created (the equivalent of using the **DirectDrawCreateClipper** function).

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_ALREADYINITIALIZED

DDERR_INVALIDPARAMS

This method is provided for compliance with the Component Object Model (COM) protocol. If **DirectDrawCreateClipper** or the **IDirectDraw2::CreateClipper** method was used to create the DirectDrawClipper object, this method returns DDERR_ALREADYINITIALIZED.

Remarks

For more information about using **IDirectDrawClipper::Initialize** with **CoCreateInstance**, see Creating DirectDrawClipper Objects with CoCreateInstance.

See Also

IUnknown::AddRef, **IUnknown::QueryInterface**, **IUnknown::Release**, **IDirectDraw2::CreateClipper**

IDirectDrawClipper::IsClipListChanged

The **IDirectDrawClipper::IsClipListChanged** method monitors the status of the clip list if a window handle is associated with a DirectDrawClipper object.

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

Parameters

lpbChanged
Address of a variable that is set to TRUE if the clip list has changed.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

IDirectDrawClipper::SetClipList

The **IDirectDrawClipper::SetClipList** method sets or deletes the clip list used by the **IDirectDrawSurface3::Blt**, **IDirectDrawSurface3::BltBatch**, and **IDirectDrawSurface3::UpdateOverlay** methods on surfaces to which the parent DirectDrawClipper object is attached.

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

Parameters

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

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CLIPPERISUSINGHWND
DDERR_INVALIDCLIPLIST
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_OUTOFMEMORY

Remarks

The clip list cannot be set if a window handle is already associated with the DirectDrawClipper object. Note that the **IDirectDrawSurface3::BltFast** method cannot clip.

The **RGNDATA** structure used with this method has the following syntax.

```
typedef struct _RGNDATA {
    RGNDATAHEADER rdh;
    char          Buffer[1];
} RGNDATA;
```

The **rdh** member of the **RGNDATA** structure is an **RGNDATAHEADER** structure that has the following syntax.

```
typedef struct _RGNDATAHEADER {
    DWORD dwSize;
    DWORD iType;
    DWORD nCount;
    DWORD nRgnSize;
    RECT  rcBound;
} RGNDATAHEADER;
```

For more information about these structures, see the documentation in the Platform Software Development Kit.

See Also

IDirectDrawClipper::GetClipList, **IDirectDrawSurface3::Blt**, **IDirectDrawSurface3::BltFast**, **IDirectDrawSurface3::BltBatch**, **IDirectDrawSurface3::UpdateOverlay**

IDirectDrawClipper::SetHWnd

The **IDirectDrawClipper::SetHWnd** method sets the window handle that will obtain the clipping information.

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

Parameters

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

hWnd
Window handle that obtains the clipping information.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDCLIPLIST
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_OUTOFMEMORY

See Also

IDirectDrawClipper::GetHWnd

IDirectDrawColorControl

The **IDirectDrawColorControl** interface allows you to get and set color controls:

Color controls	**GetColorControls**
	SetColorControls

The **IDirectDrawColorControl** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface

Release

You can use the LPDIRECTDRAWCOLORCONTROL data type to declare a variable that contains a pointer to an **IDirectDrawColorControl** interface. The Ddraw.h header file declares these data types with the following code:

```
typedef struct IDirectDrawColorControl    FAR *LPDIRECTDRAWCOLORCONTROL;
```

IDirectDrawColorControl::GetColorControls

The **IDirectDrawColorControl::GetColorControls** method returns the current color control settings associated with the specified overlay or primary surface. The dwFlags member of the **DDCOLORCONTROL** structure indicate which of the color control options are supported.

```
HRESULT GetColorControls(
  LPDDCOLORCONTROL lpColorControl
);
```

Parameters

lpColorControl
Address of the **DDCOLORCONTROL** structure that will receive the current control settings of the specified surface.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

IDirectDrawColorControl::SetColorControls

The **IDirectDrawColorControl::SetColorControls** method sets the color control settings associated with the specified overlay or primary surface.

```
HRESULT SetColorControls(
  LPDDCOLORCONTROL lpColorControl
);
```

Parameters

lpColorControl
Address of the **DDCOLORCONTROL** structure containing the new values to be applied to the specified surface.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

IDirectDrawPalette

Applications use the methods of the **IDirectDrawPalette** interface to create DirectDrawPalette objects and work with system-level variables. This section is a reference to the methods of this interface. For a conceptual overview, see Palettes.

The methods of the **IDirectDrawPalette** interface can be organized into the following groups:

Allocating memory	**Initialize**
Palette capabilities	**GetCaps**
Palette entries	**GetEntries** **SetEntries**

The **IDirectDrawPalette** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef
QueryInterface
Release

You can use the LPDIRECTDRAWPALETTE data type to declare a variable that contains a pointer to an **IDirectDrawPalette** interface. The Ddraw.h header file declares these data types with the following code:

```
typedef struct IDirectDrawPalette    FAR *LPDIRECTDRAWPALETTE;
```

IDirectDrawPalette::GetCaps

The **IDirectDrawPalette::GetCaps** method retrieves the capabilities of this palette object.

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

Parameters

lpdwCaps
Flag from the **dwPalCaps** member of the **DDCAPS** structure that defines palette capabilities:

DDPCAPS_1BIT
DDPCAPS_2BIT
DDPCAPS_4BIT
DDPCAPS_8BIT
DDPCAPS_8BITENTRIES
DDPCAPS_ALLOW256
DDPCAPS_PRIMARYSURFACE
DDPCAPS_PRIMARYSURFACELEFT
DDPCAPS_VSYNC

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

IDirectDrawPalette::GetEntries

The **IDirectDrawPalette::GetEntries** method queries palette values from a DirectDrawPalette object.

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

Parameters

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

dwBase
Start of the entries that should be retrieved sequentially.

dwNumEntries
Number of palette entries that can fit in the address specified in *lpEntries*. The colors of each palette entry are returned in sequence, from the value of the *dwStartingEntry* parameter through the value of the *dwCount* parameter minus 1. (These parameters are set by **IDirectDrawPalette::SetEntries**.)

lpEntries
Address of the palette entries. The palette entries are 1 byte each if the DDPCAPS_8BITENTRIES flag is set and 4 bytes otherwise. Each field is a color description.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_NOTPALETTIZED

See Also

IDirectDrawPalette::SetEntries

IDirectDrawPalette::Initialize

The **IDirectDrawPalette::Initialize** method initializes the DirectDrawPalette object.

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

Parameters

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

dwFlags and *lpDDColorTable*
These parameters are currently not used and must be set to 0.

Return Values

This method returns DDERR_ALREADYINITIALIZED.

This method is provided for compliance with the Component Object Model (COM) protocol. Because the DirectDrawPalette object is initialized when it is created, this method always returns DDERR_ALREADYINITIALIZED.

See Also

IUnknown::AddRef, **IUnknown::QueryInterface**, **IUnknown::Release**

IDirectDrawPalette::SetEntries

The **IDirectDrawPalette::SetEntries** method changes entries in a DirectDrawPalette object immediately.

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

Parameters

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

dwStartingEntry
First entry to be set.

dwCount
Number of palette entries to be changed.

lpEntries
Address of the palette entries. The palette entries are 1 byte each if the DDPCAPS_8BITENTRIES flag is set and 4 bytes otherwise. Each field is a color description.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_NOPALETTEATTACHED

DDERR_NOTPALETTIZED
DDERR_UNSUPPORTED

See Also

IDirectDrawPalette::GetEntries, **IDirectDrawSurface3::SetPalette**

IDirectDrawSurface3

Applications use the methods of the **IDirectDrawSurface3** interface to create DirectDrawSurface objects and work with system-level variables. This section is a reference to the methods of this interface. For a conceptual overview, see Surfaces.

The methods of the **IDirectDrawSurface3** interface can be organized into the following groups:

Allocating memory	**Initialize**
	IsLost
	Restore
Attaching surfaces	**AddAttachedSurface**
	DeleteAttachedSurface
	EnumAttachedSurfaces
	GetAttachedSurface
Blitting	**Blt**
	BltBatch
	BltFast
	GetBltStatus
Color	**GetColorKey**
	SetColorKey
Device contexts	**GetDC**
	ReleaseDC
Flipping	**Flip**
	GetFlipStatus
Locking surfaces	**Lock**

	PageLock
	PageUnlock
	Unlock
Miscellaneous	**GetDDInterface**
Overlays	**AddOverlayDirtyRect**
	EnumOverlayZOrders
	GetOverlayPosition
	SetOverlayPosition
	UpdateOverlay
	UpdateOverlayDisplay
	UpdateOverlayZOrder
Surface capabilities	**GetCaps**
Surface clipper	**GetClipper**
	SetClipper
Surface description	**GetPixelFormat**
	GetSurfaceDesc
	SetSurfaceDesc
Surface palettes	**GetPalette**
	SetPalette

The **IDirectDrawSurface3** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef

QueryInterface

Release

You can use the LPDIRECTDRAWSURFACE, LPDIRECTDRAWDURFACE2, or LPDIRECTDRAWSURFACE3 data types to declare variables that point to an **IDirectDrawSurface**, **IDirectDrawSurface2**, or **IDirectDrawSurface3** interface. The Ddraw.h header file declares these data types with the following code:

```
typedef struct IDirectDrawSurface     FAR *LPDIRECTDRAWSURFACE;
typedef struct IDirectDrawSurface2    FAR *LPDIRECTDRAWSURFACE2;
```

typedef struct IDirectDrawSurface3 FAR *LPDIRECTDRAWSURFACE3;

IDirectDrawSurface3::AddAttachedSurface

The **IDirectDrawSurface3::AddAttachedSurface** method attaches a surface to another surface.

HRESULT AddAttachedSurface(
LPDIRECTDRAWSURFACE3 *lpDDSAttachedSurface*
);

Parameters

lpDDSAttachedSurface
Address of the DirectDraw surface that is to be attached.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CANNOTATTACHSURFACE
DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACEALREADYATTACHED
DDERR_SURFACELOST
DDERR_WASSTILLDRAWING

Remarks

Possible attachments include z-buffers, alpha channels, and back buffers. Some attachments automatically break other attachments. For example, the 3-D z-buffer can be attached only to one back buffer at a time. Attachment is not bidirectional, and a surface cannot be attached to itself. Emulated surfaces (in system memory) cannot be attached to nonemulated surfaces. Unless one surface is a texture map, the two attached surfaces must be the same size. A flipping surface cannot be attached to another flipping surface of the same type; however, attaching two surfaces of different types is allowed. For example, a flipping z-buffer can be attached to a regular flipping surface. If a nonflipping surface is attached to another nonflipping surface of the same type, the two surfaces will become a flipping chain. If a nonflipping surface is attached to a flipping surface, it becomes part of the existing

flipping chain. Additional surfaces can be added to this chain, and each call of the **IDirectDrawSurface3::Flip** method will advance one step through the surfaces.

See Also

IDirectDrawSurface3::DeleteAttachedSurface,
IDirectDrawSurface3::EnumAttachedSurfaces

IDirectDrawSurface3::AddOverlayDirtyRect

This method is not currently implemented.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE
DDERR_UNSUPPORTED

See Also

IDirectDrawSurface3::UpdateOverlayDisplay

IDirectDrawSurface3::Blt

The **IDirectDrawSurface3::Blt** method performs a bit block transfer. This method (as well as the **IDirectDrawSurface3** version) does not currently support z-buffering or alpha blending (see alpha channel) during blit operations.

```
HRESULT Blt(
  LPRECT lpDestRect,
  LPDIRECTDRAWSURFACE3 lpDDSrcSurface,
```

LPRECT *lpSrcRect*,
DWORD *dwFlags*,
LPDDBLTFX *lpDDBltFx*
);

Parameters

lpDestRect
Address of a **RECT** structure that defines the upper-left and lower-right points of the rectangle on the destination surface to be blitted to. If this parameter is NULL, the entire destination surface will be used.

lpDDSrcSurface
Address of the DirectDraw surface that is the source for the blit operation.

lpSrcRect
Address of a **RECT** structure that defines the upper-left and lower-right points of the rectangle on the source surface to be blitted from. If this parameter is NULL, the entire source surface will be used.

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

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

DDBLT_ALPHADESTNEG
Indicates that the destination surface becomes more transparent as the alpha value increases (0 is opaque).

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

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

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

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

DDBLT_ALPHASRCNEG

Indicates that the source surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHASRCSURFACEOVERRIDE

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

DDBLT_ASYNC

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

DDBLT_COLORFILL

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

DDBLT_DDFX

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

DDBLT_DDROPS

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

DDBLT_DEPTHFILL

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

DDBLT_KEYDEST

Uses the color key associated with the destination surface.

DDBLT_KEYDESTOVERRIDE

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

DDBLT_KEYSRC

Uses the color key associated with the source surface.

DDBLT_KEYSRCOVERRIDE

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

DDBLT_ROP

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

DDBLT_ROTATIONANGLE

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

DDBLT_WAIT

Postpones the DDERR_WASSTILLDRAWING return value if the blitter is busy, and returns as soon as the blit can be set up or another

error occurs.

DDBLT_ZBUFFER

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

DDBLT_ZBUFFERDESTCONSTOVERRIDE

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

DDBLT_ZBUFFERDESTOVERRIDE

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

DDBLT_ZBUFFERSRCCONSTOVERRIDE

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

DDBLT_ZBUFFERSRCOVERRIDE

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

lpDDBltFx

Address of the **DDBLTFX** structure.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDCLIPLIST
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDRECT
DDERR_NOALPHAHW
DDERR_NOBLTHW
DDERR_NOCLIPLIST
DDERR_NODDROPSHW
DDERR_NOMIRRORHW
DDERR_NORASTEROPHW
DDERR_NOROTATIONHW

DDERR_NOSTRETCHHW
DDERR_NOZBUFFERHW
DDERR_SURFACEBUSY
DDERR_SURFACELOST
DDERR_UNSUPPORTED

Remarks

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

Typically, **IDirectDrawSurface3::Blt** returns immediately with an error if the blitter is busy and the blit could not be set up. Specify the DDBLT_WAIT flag to request a synchronous blit. When you include the DDBLT_WAIT flag, the method waits until the blit can be set up or another error occurs before it returns.

Note that **RECT** structures are defined so that the **right** and **bottom** members are exclusive—therefore, **right** – **left** equals the width of the rectangle, not one less than the width.

IDirectDrawSurface3::BltBatch

The **IDirectDrawSurface3::BltBatch** method performs a sequence of **IDirectDrawSurface3::Blt** operations from several sources to a single destination. This method is currently only a stub; it has not yet been implemented.

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

Parameters

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

dwCount
Number of blit operations to be performed.

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

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

IDirectDrawSurface3::BltFast

The **IDirectDrawSurface3::BltFast** method performs a source copy blit or transparent blit by using a source color key or destination color key.

```
HRESULT BltFast(
  DWORD dwX,
  DWORD dwY,
  LPDIRECTDRAWSURFACE3 lpDDSrcSurface,
  LPRECT lpSrcRect,
  DWORD dwTrans
);
```

Parameters

dwX and *dwY*
The x- and y-coordinates to blit to on the destination surface.

lpDDSrcSurface

Address of the DirectDraw surface that is the source for the blit operation.

lpSrcRect

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

dwTrans

Type of transfer.

DDBLTFAST_DESTCOLORKEY

Specifies a transparent blit that uses the destination's color key.

DDBLTFAST_NOCOLORKEY

Specifies a normal copy blit with no transparency.

DDBLTFAST_SRCCOLORKEY

Specifies a transparent blit that uses the source's color key.

DDBLTFAST_WAIT

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_EXCEPTION
DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDRECT
DDERR_NOBLTHW
DDERR_SURFACEBUSY
DDERR_SURFACELOST
DDERR_UNSUPPORTED

Remarks

This method always attempts an asynchronous blit if it is supported by the hardware.

This method works only on display memory surfaces and cannot clip when blitting. If you use this method on a surface with an attached clipper, the call will fail and the method will return DDERR_UNSUPPORTED.

The software implementation of **IDirectDrawSurface3::BltFast** is 10 percent faster than the **IDirectDrawSurface3::Blt** method. However, there is no speed difference between the two if display hardware is being used.

Typically, **IDirectDrawSurface3::BltFast** returns immediately with an error if the blitter is busy and the blit cannot be set up. You can use the DDBLTFAST_WAIT flag, however, if you want this method to not return until either the blit can be set up or another error occurs.

IDirectDrawSurface3::DeleteAttachedSurface

The **IDirectDrawSurface3::DeletAttachedSurface** method detaches two attached surfaces. The detached surface is not released.

```
HRESULT DeleteAttachedSurface(
  DWORD dwFlags,
  LPDIRECTDRAWSURFACE3 lpDDSAttachedSurface
);
```

Parameters

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

lpDDSAttachedSurface
Address of the DirectDraw surface to be detached. If this parameter is NULL, all attached surfaces are detached.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CANNOTDETACHSURFACE
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACELOST
DDERR_SURFACENOTATTACHED

Remarks

Implicit attachments, those formed by DirectDraw rather than the **IDirectDrawSurface3::AddAttachedSurface** method, cannot be detached. Detaching surfaces from a flipping chain can alter other surfaces in the chain. If a front buffer is detached from a flipping chain, the next surface in the chain becomes the front buffer, and the following surface becomes the back buffer. If a back buffer is detached from a chain, the following surface becomes a back buffer. If a plain surface is detached from a chain, the chain simply becomes shorter. If a flipping chain has

only two surfaces and they are detached, the chain is destroyed and both surfaces return to their previous designations.

See Also

IDirectDrawSurface3::Flip

IDirectDrawSurface3::EnumAttachedSurfaces

The **IDirectDrawSurface3::EnumAttachedSurfaces** method enumerates all the surfaces attached to a given surface.

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

Parameters

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

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACELOST

IDirectDrawSurface3::EnumOverlayZOrders

The **IDirectDrawSurface3::EnumOverlayZOrders** method enumerates the overlay surfaces on the specified destination. The overlays can be enumerated in front-to-back or back-to-front order.

```
HRESULT EnumOverlayZOrders(
```

```
  DWORD dwFlags,
  LPVOID lpContext,
  LPDDENUMSURFACESCALLBACK lpfnCallback
);
```

Parameters

dwFlags

One of the following flags:

DDENUMOVERLAYZ_BACKTOFRONT

Enumerates overlays back to front.

DDENUMOVERLAYZ_FRONTTOBACK

Enumerates overlays front to back.

lpContext

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

lpfnCallback

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

IDirectDrawSurface3::Flip

The **IDirectDrawSurface3::Flip** method makes the surface memory associated with the DDSCAPS_BACKBUFFER surface become associated with the front-buffer surface.

```
HRESULT Flip(
  LPDIRECTDRAWSURFACE3 lpDDSurfaceTargetOverride,
  DWORD dwFlags
);
```

Parameters

lpDDSurfaceTargetOverride

Address of another surface in the flipping chain that will be flipped to. The specified surface must be a member of the flipping chain. The default for this parameter is NULL, in which case DirectDraw cycles through the buffers in the order they are attached to each other.

dwFlags

Flags specifying flip options.

DDFLIP_EVEN

For use only when displaying video in an overlay surface. The new surface contains data from the even field of a video signal. This flag cannot be used with the DDFLIP_ODD flag.

DDFLIP_ODD

For use only when displaying video in an overlay surface. The new surface contains data from the odd field of a video signal. This flag cannot be used with the DDFLIP_EVEN flag.

DDFLIP_WAIT

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOFLIPHW
DDERR_NOTFLIPPABLE
DDERR_SURFACEBUSY
DDERR_SURFACELOST
DDERR_UNSUPPORTED
DDERR_WASSTILLDRAWING

Remarks

This method can be called only for a surface that has the DDSCAPS_FLIP and DDSCAPS_FRONTBUFFER capabilities. The display memory previously associated with the front buffer is associated with the back buffer.

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

The **IDirectDrawSurface3::Flip** method will always be synchronized with the vertical blank. If the surface has been assigned to a video port, this method updates the visible overlay surface and the video port's target surface.

For more information, see Flipping Surfaces.

See Also

IDirectDrawSurface3::GetFlipStatus

IDirectDrawSurface3::GetAttachedSurface

The **IDirectDrawSurface3::GetAttachedSurface** method obtains the attached surface that has the specified capabilities.

```
HRESULT GetAttachedSurface(
  LPDDSCAPS lpDDSCaps,
  LPDIRECTDRAWSURFACE3 FAR *lplpDDAttachedSurface
);
```

Parameters

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

lplpDDAttachedSurface
Address of a variable that will contain a pointer to the retrieved surface's **IDirectDrawSurface3** interface. The retrieved surface is the one that matches the description according to the *lpDDSCaps* parameter.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS
DDERR_NOTFOUND
DDERR_SURFACELOST

Remarks

Attachments are used to connect multiple DirectDrawSurface objects into complex structures, like the ones needed to support 3-D page flipping with z-buffers. This method fails if more than one surface is attached that matches the capabilities requested. In this case, the application must use the **IDirectDrawSurface3::EnumAttachedSurfaces** method to obtain the attached surfaces.

IDirectDrawSurface3::GetBltStatus

The **IDirectDrawSurface3::GetBltStatus** method obtains the blitter status.

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

Parameters

dwFlags
One of the following flags:

DDGBS_CANBLT
Inquires whether a blit involving this surface can occur immediately, and returns DD_OK if the blit can be completed.

DDGBS_ISBLTDONE
Inquires whether the blit is done, and returns DD_OK if the last blit on this surface has completed.

Return Values

If the method succeeds, that means a blitter is present, the return value is DD_OK.

If the method fails, the return value is DDERR_WASSTILLDRAWING if the blitter is busy, DDERR_NOBLTHW if there is no blitter, or one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOBLTHW
DDERR_SURFACEBUSY
DDERR_SURFACELOST

DDERR_UNSUPPORTED
DDERR_WASSTILLDRAWING

IDirectDrawSurface3::GetCaps

The **IDirectDrawSurface3::GetCaps** method retrieves the capabilities of the surface. These capabilities are not necessarily related to the capabilities of the display device.

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

Parameters

lpDDSCaps
Address of a **DDSCAPS** structure that will be filled with the hardware capabilities of the surface.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

IDirectDrawSurface3::GetClipper

The **IDirectDrawSurface3::GetClipper** method retrieves the DirectDrawClipper object associated with this surface.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOCLIPPERATTACHED

See Also

IDirectDrawSurface3::SetClipper

IDirectDrawSurface3::GetColorKey

The **IDirectDrawSurface3::GetColorKey** method retrieves the color key value for the DirectDrawSurface object.

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

Parameters

dwFlags
Determines which color key is requested.

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

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

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

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

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOCOLORKEY
DDERR_NOCOLORKEYHW
DDERR_SURFACELOST
DDERR_UNSUPPORTED

See Also

IDirectDrawSurface3::SetColorKey

IDirectDrawSurface3::GetDC

The **IDirectDrawSurface3::GetDC** method creates a GDI-compatible handle of a device context for the surface.

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

Parameters

lphDC
Address for the returned handle to a device context.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_DCALREADYCREATED
DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE
DDERR_SURFACELOST
DDERR_UNSUPPORTED
DDERR_WASSTILLDRAWING

Remarks

This method uses an internal version of the **IDirectDrawSurface3::Lock** method to lock the surface. The surface remains locked until the **IDirectDrawSurface3::ReleaseDC** method is called.

See Also

IDirectDrawSurface3::Lock

IDirectDrawSurface3::GetDDInterface

The **IDirectDrawSurface3::GetDDInterface** method retrieves an interface to the DirectDraw object that was used to create the surface.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method was not implemented in the **IDirectDraw** interface.

IDirectDrawSurface3::GetFlipStatus

The **IDirectDrawSurface3::GetFlipStatus** method indicates whether the surface has finished its flipping process.

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

Parameters

dwFlags
One of the following flags:

DDGFS_CANFLIP
Inquires whether this surface can be flipped immediately and returns DD_OK if the flip can be completed.

DDGFS_ISFLIPDONE
Inquires whether the flip has finished and returns DD_OK if the last flip on this surface has completed.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is DDERR_WASSTILLDRAWING if the surface has not finished its flipping process, or one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE
DDERR_SURFACEBUSY
DDERR_SURFACELOST
DDERR_UNSUPPORTED

See Also

IDirectDrawSurface3::Flip

IDirectDrawSurface3::GetOverlayPosition

Given a visible, active overlay surface (DDSCAPS_OVERLAY flag set), the **IDirectDrawSurface3::GetOverlayPosition** method returns the display coordinates of the surface.

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

Parameters

lplX and *lplY*
Addresses of the x- and y-display coordinates.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDPOSITION
DDERR_NOOVERLAYDEST
DDERR_NOTAOVERLAYSURFACE
DDERR_OVERLAYNOTVISIBLE
DDERR_SURFACELOST

See Also

IDirectDrawSurface3::SetOverlayPosition,
IDirectDrawSurface3::UpdateOverlay

IDirectDrawSurface3::GetPalette

The **IDirectDrawSurface3::GetPalette** method retrieves the DirectDrawPalette structure associated with this surface and increments the reference count of the returned palette.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS
DDERR_NOEXCLUSIVEMODE
DDERR_NOPALETTEATTACHED
DDERR_SURFACELOST
DDERR_UNSUPPORTED

See Also

IDirectDrawSurface3::SetPalette

IDirectDrawSurface3::GetPixelFormat

The **IDirectDrawSurface3::GetPixelFormat** method retrieves the color and pixel format of the surface.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE

IDirectDrawSurface3::GetSurfaceDesc

The **IDirectDrawSurface3::GetSurfaceDesc** method retrieves a **DDSURFACEDESC** structure that describes the surface in its current condition.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

DDSURFACEDESC

IDirectDrawSurface3::Initialize

The **IDirectDrawSurface3::Initialize** method initializes a DirectDrawSurface object.

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

Parameters

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

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

Return Values

The method returns DDERR_ALREADYINITIALIZED.

Remarks

This method is provided for compliance with the Component Object Model (COM) protocol. Because the DirectDrawSurface object is initialized when it is created, this method always returns DDERR_ALREADYINITIALIZED.

See Also

IUnknown::AddRef, **IUnknown::QueryInterface**, **IUnknown::Release**

IDirectDrawSurface3::IsLost

The **IDirectDrawSurface3::IsLost** method determines if the surface memory associated with a DirectDrawSurface object has been freed.

HRESULT IsLost();

Return Values

If the method succeeds, the return value is DD_OK because the memory has not been freed.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_SURFACELOST

You can use this method to reallocate surface memory. When a DirectDrawSurface object loses its surface memory, most methods return DDERR_SURFACELOST and perform no other action.

Remarks

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

See Also

IDirectDrawSurface3::Restore

IDirectDrawSurface3::Lock

The **IDirectDrawSurface3::Lock** method obtains a pointer to the surface memory.

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

Parameters

lpDestRect
Address of a **RECT** structure that identifies the region of surface that is being locked. If NULL, the entire surface will be locked.

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

dwFlags

DDLOCK_EVENT
This flag is not currently implemented.

DDLOCK_NOSYSLOCK
If possible, do not take the Win16Lock. This flag is ignored when locking the primary surface.

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

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

DDLOCK_WAIT
If a lock cannot be obtained because a blit operation is in progress, the method retries until a lock is obtained or another error occurs, such as DDERR_SURFACEBUSY.

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

hEvent
This parameter is not used and must be set to NULL.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_OUTOFMEMORY
DDERR_SURFACEBUSY
DDERR_SURFACELOST
DDERR_WASSTILLDRAWING

Remarks

For more information on using this method, see Accessing the Frame-Buffer Directly.

After retrieving a surface memory pointer, you can access the surface memory until a corresponding **IDirectDrawSurface3::Unlock** method is called. When the surface is unlocked, the pointer to the surface memory is invalid.

Do not call DirectDraw blit functions to blit from a locked region of a surface. If you do, the blit returns either DDERR_SURFACEBUSY or DDERR_LOCKEDSURFACES. Additionally, GDI blit functions will silently fail when used on a locked video memory surface.

This method often causes DirectDraw to hold the Win16Lock until you call the **IDirectDrawSurface3::Unlock** method. GUI debuggers cannot operate while the Win16Lock is held.

See Also

IDirectDrawSurface3::Unlock, **IDirectDrawSurface3::GetDC**, **IDirectDrawSurface3::ReleaseDC**

IDirectDrawSurface3::PageLock

The **IDirectDrawSurface3::PageLock** method prevents a system-memory surface from being paged out while a blit operation using direct memory access (DMA) transfers to or from system memory is in progress.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CANTPAGELOCK

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_SURFACELOST

Remarks

You must call this method to make use of DMA support. If you do not, the blit occurs using software emulation. For more information, see Using DMA.

The performance of the operating system could be negatively affected if too much memory is locked.

A lock count is maintained for each surface and is incremented each time **IDirectDrawSurface3::PageLock** is called for that surface. The count is decremented when **IDirectDrawSurface3::PageUnlock** is called. When the count reaches 0, the memory is unlocked and can then be paged by the operating system.

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

This method was not implemented in the **IDirectDraw** interface.

See Also

IDirectDrawSurface3::PageUnlock

IDirectDrawSurface3::PageUnlock

The **IDirectDrawSurface3::PageUnlock** method unlocks a system-memory surface, allowing it to be paged out.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_CANTPAGEUNLOCK
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOTPAGELOCKED
DDERR_SURFACELOST

Remarks

A lock count is maintained for each surface and is incremented each time **IDirectDrawSurface3::PageLock** is called for that surface. The count is decremented when **IDirectDrawSurface3::PageUnlock** is called. When the count reaches 0, the memory is unlocked and can then be paged by the operating system.

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

This method was not implemented in the **IDirectDraw** interface.

See Also

IDirectDrawSurface3::PageLock

IDirectDrawSurface3::ReleaseDC

The **IDirectDrawSurface3::ReleaseDC** method releases the handle of a device context previously obtained by using the **IDirectDrawSurface3::GetDC** method.

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

Parameters

hDC
Handle to a device context previously obtained by **IDirectDrawSurface3::GetDC**.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACELOST
DDERR_UNSUPPORTED

Remarks

This method also unlocks the surface previously locked when the **IDirectDrawSurface3::GetDC** method was called.

See Also

IDirectDrawSurface3::GetDC

IDirectDrawSurface3::Restore

The **IDirectDrawSurface3::Restore** method restores a surface that has been lost. This occurs when the surface memory associated with the DirectDrawSurface object has been freed.

HRESULT Restore();

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_IMPLICITLYCREATED
DDERR_INCOMPATIBLEPRIMARY
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_NOEXCLUSIVEMODE
DDERR_OUTOFMEMORY
DDERR_UNSUPPORTED
DDERR_WRONGMODE

Remarks

This method restores the memory allocated for a surface, but not the contents of that memory. The application is responsible for restoring the contents to reestablish any lost images.

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 card. When a DirectDrawSurface object loses its surface memory, many methods will return DDERR_SURFACELOST and perform no other function. The **IDirectDrawSurface3::Restore** method will reallocate surface memory and reattach it to the DirectDrawSurface object.

A single call to this method will restore a DirectDrawSurface object's associated implicit surfaces (back buffers, and so on). An attempt to restore an implicitly created

surface will result in an error. **IDirectDrawSurface3::Restore** will not work across explicit attachments created by using the **IDirectDrawSurface3::AddAttachedSurface** method — each of these surfaces must be restored individually.

See Also

IDirectDrawSurface3::IsLost, **IDirectDrawSurface3::AddAttachedSurface**

IDirectDrawSurface3::SetClipper

The **IDirectDrawSurface3::SetClipper** method attaches a DirectDrawClipper object to a DirectDrawSurface object.

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

Parameters

lpDDClipper
Address of the DirectDrawClipper structure representing the DirectDrawClipper object that will be attached to the DirectDrawSurface object. If this parameter is NULL, the current DirectDrawClipper object will be detached.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE
DDERR_NOCLIPPERATTACHED

Remarks

This method is primarily used by surfaces that are being overlaid on or blitted to the primary surface. However, it can be used on any surface. After a DirectDrawClipper object has been attached and a clip list is associated with it, the DirectDrawClipper object will be used for the **IDirectDrawSurface3::Blt**, **IDirectDrawSurface3::BltBatch**, and **IDirectDrawSurface3::UpdateOverlay** operations involving the parent DirectDrawSurface object. This method can also detach a DirectDrawSurface object's current DirectDrawClipper object.

If this method is called several times consecutively on the same surface for the same DirectDrawClipper object, the reference count for the object is incremented only once. Subsequent calls do not affect the object's reference count.

See Also

IDirectDrawSurface3::GetClipper

IDirectDrawSurface3::SetColorKey

The **IDirectDrawSurface3::SetColorKey** method sets the color key value for the DirectDrawSurface object if the hardware supports color keys on a per surface basis.

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

Parameters

dwFlags

Determines which color key is requested.

DDCKEY_COLORSPACE

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

DDCKEY_DESTBLT

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

DDCKEY_DESTOVERLAY

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

DDCKEY_SRCBLT

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

DDCKEY_SRCOVERLAY

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

lpDDColorKey

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDSURFACETYPE
DDERR_NOOVERLAYHW
DDERR_NOTAOVERLAYSURFACE
DDERR_SURFACELOST
DDERR_UNSUPPORTED
DDERR_WASSTILLDRAWING

Remarks

For transparent blits and overlays, you should set destination color on the destination surface and source color on the source surface.

See Also

IDirectDrawSurface3::GetColorKey

IDirectDrawSurface3::SetOverlayPosition

The **IDirectDrawSurface3::SetOverlayPosition** method changes the display coordinates of an overlay surface.

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

Parameters

lX and *lY*
New x- and y-display coordinates.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDPOSITION
DDERR_NOOVERLAYDEST
DDERR_NOTAOVERLAYSURFACE
DDERR_OVERLAYNOTVISIBLE
DDERR_SURFACELOST
DDERR_UNSUPPORTED

See Also

IDirectDrawSurface3::GetOverlayPosition,
IDirectDrawSurface3::UpdateOverlay

IDirectDrawSurface3::SetPalette

The **IDirectDrawSurface3::SetPalette** method attaches the specified DirectDrawPalette object to a surface. The surface uses this palette for all subsequent operations. The palette change takes place immediately, without regard to refresh timing.

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

Parameters

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDPIXELFORMAT
DDERR_INVALIDSURFACETYPE
DDERR_NOEXCLUSIVEMODE

DDERR_NOPALETTEATTACHED
DDERR_NOPALETTEHW
DDERR_NOT8BITCOLOR
DDERR_SURFACELOST
DDERR_UNSUPPORTED

Remarks

If this method is called several times consecutively on the same surface for the same palette, the reference count for the palette is incremented only once. Subsequent calls do not affect the palette's reference count.

See Also

IDirectDrawSurface3::GetPalette, **IDirectDraw2::CreatePalette**

IDirectDrawSurface3::SetSurfaceDesc

The **IDirectDrawSurface3::SetSurfaceDesc** method sets the characteristics of an existing surface. This method is new with the **IDirectDrawSurface3** interface.

```
HRESULT IDirectDrawSurface3::SetSurfaceDesc(
  LPDDSURFACEDESC lpddsd,
  DWORD dwFlags
);
```

Parameters

lpddsd
Address of a **DDSURFACEDESC** structure that contains the new surface characteristics.

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDPARAMS
DDERR_INVALIDOBJECT
DDERR_SURFACELOST
DDERR_SURFACEBUSY
DDERR_INVALIDSURFACETYPE

DDERR_INVALIDPIXELFORMAT
DDERR_INVALIDCAPS
DDERR_UNSUPPORTED
DDERR_GENERIC

Remarks

Currently, this method can only be used to set the surface data and pixel format used by an explicit system memory surface. This is useful as it allows a surface to use data from a previously allocated buffer without copying. The new surface memory is allocated by the client application and, as such, the client application must also deallocate it. For more information about how this method is used, see Updating Surface Characteristics.

Using this method incorrectly will cause unpredictable behavior. The DirectDrawSurface object will not deallocate surface memory that it didn't allocate. Therefore, when the surface memory is no longer needed, it is your responsibility to deallocate it. However, when this method is called, DirectDraw frees the original surface memory that it implicitly allocated when creating the surface.

IDirectDrawSurface3::Unlock

The **IDirectDrawSurface3::Unlock** method notifies DirectDraw that the direct surface manipulations are complete.

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

Parameters

lpSurfaceData
Address of the surface to be unlocked, as retrieved by the **IDirectDrawSurface3::Lock** method. This parameter can be NULL only if the entire surface was locked by passing NULL in the *lpDestRect* parameter of the corresponding call to the **IDirectDrawSurface3::Lock** method.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_GENERIC
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

DDERR_INVALIDRECT
DDERR_NOTLOCKED
DDERR_SURFACELOST

Remarks

Because it is possible to call **IDirectDrawSurface3::Lock** multiple times for the same surface with different destination rectangles, the pointer in *lpSurfaceData* links the calls to the **IDirectDrawSurface3::Lock** and **IDirectDrawSurface3::Unlock** methods.

See Also

IDirectDrawSurface3::Lock

IDirectDrawSurface3::UpdateOverlay

The **IDirectDrawSurface3::UpdateOverlay** method repositions or modifies the visual attributes of an overlay surface. These surfaces must have the DDSCAPS_OVERLAY value set.

```
HRESULT UpdateOverlay(
  LPRECT lpSrcRect,
  LPDIRECTDRAWSURFACE3 lpDDDestSurface,
  LPRECT lpDestRect,
  DWORD dwFlags,
  LPDDOVERLAYFX lpDDOverlayFx
);
```

Parameters

lpSrcRect
Address of a **RECT** structure that defines the x, y, width, and height of the region on the source surface being used as the overlay. This parameter can be NULL when hiding an overlay or to indicate that the entire overlay surface is to be used and that the overlay surface conforms to any boundary and size alignment restrictions imposed by the device driver.

lpDDDestSurface
Address of the DirectDraw surface that is being overlaid.

lpDestRect
Address of a **RECT** structure that defines the x, y, width, and height of the region on the destination surface that the overlay should be moved to. This parameter can be NULL when hiding the overlay.

dwFlags
DDOVER_ADDDIRTYRECT

Adds a dirty rectangle to an emulated overlaid surface.

DDOVER_ALPHADEST

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

DDOVER_ALPHADESTCONSTOVERRIDE

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

DDOVER_ALPHADESTNEG

Indicates that the destination surface becomes more transparent as the alpha value increases (0 is opaque).

DDOVER_ALPHADESTSURFACEOVERRIDE

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

DDOVER_ALPHAEDGEBLEND

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

DDOVER_ALPHASRC

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

DDOVER_ALPHASRCCONSTOVERRIDE

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

DDOVER_ALPHASRCNEG

Indicates that the source surface becomes more transparent as the alpha value increases (0 is opaque).

DDOVER_ALPHASRCSURFACEOVERRIDE

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

DDOVER_AUTOFLIP

Automatically flip to the next surface in the flip chain each time a video port VSYNC occurs.

DDOVER_BOB

Display each field individually of the interlaced video stream without causing any artifacts.

DDOVER_DDFX

Uses the overlay FX flags in the *lpDDOverlayFx* parameter to define special overlay effects.

DDOVER_HIDE

Turns this overlay off.

DDOVER_KEYDEST

Uses the color key associated with the destination surface.

DDOVER_KEYDESTOVERRIDE

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

DDOVER_KEYSRC

Uses the color key associated with the source surface.

DDOVER_KEYSRCOVERRIDE

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

DDOVER_OVERRIDEBOBWEAVE

Indicates that bob/weave decisions should not be overridden by other interfaces.

DDOVER_INTERLEAVED

Indicates that the surface memory is composed of interleaved fields.

DDOVER_SHOW

Turns this overlay on.

lpDDOverlayFx

Address of a **DDOVERLAYFX** structure that describes the effects to be used. This parameter can be NULL if the DDOVER_DDFX flag is not specified.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_DEVICEDOESNTOWNSURFACE
DDERR_GENERIC
DDERR_HEIGHTALIGN
DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_INVALIDRECT
DDERR_INVALIDSURFACETYPE
DDERR_NOSTRETCHHW
DDERR_NOTAOVERLAYSURFACE
DDERR_OUTOFCAPS
DDERR_SURFACELOST
DDERR_UNSUPPORTED

DDERR_XALIGN

IDirectDrawSurface3::UpdateOverlayDisplay

The **IDirectDrawSurface3::UpdateOverlayDisplay** method repaints the rectangles in the dirty rectangle list of all active overlays. This clears the dirty rectangle list. This method is for software emulation only—it does nothing if the hardware supports overlays.

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

Parameters

dwFlags

Type of update to perform. One of the following flags:

DDOVER_REFRESHDIRTYRECTS

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

DDOVER_REFRESHALL

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

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_INVALIDSURFACETYPE

DDERR_UNSUPPORTED

Remarks

This method is not currently implemented.

See Also

IDirectDrawSurface3::AddOverlayDirtyRect

IDirectDrawSurface3::UpdateOverlayZOrder

The **IDirectDrawSurface3::UpdateOverlayZOrder** method sets the z-order of an overlay.

```
HRESULT UpdateOverlayZOrder(
  DWORD dwFlags,
  LPDIRECTDRAWSURFACE3 lpDDSReference
);
```

Parameters

dwFlags

One of the following flags:

DDOVERZ_INSERTINBACKOF

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

DDOVERZ_INSERTINFRONTOF

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

DDOVERZ_MOVEBACKWARD

Moves this overlay one position backward in the overlay chain.

DDOVERZ_MOVEFORWARD

Moves this overlay one position forward in the overlay chain.

DDOVERZ_SENDTOBACK

Moves this overlay to the back of the overlay chain.

DDOVERZ_SENDTOFRONT

Moves this overlay to the front of the overlay chain.

lpDDSReference

Address of the DirectDraw surface to be used as a relative position in the overlay chain. This parameter is needed only for DDOVERZ_INSERTINBACKOF and DDOVERZ_INSERTINFRONTOF.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_NOTAOVERLAYSURFACE

See Also

IDirectDrawSurface3::EnumOverlayZOrders

IDirectDrawVideoPort

Applications use the methods of the **IDirectDrawVideoPort** interface to channel live video data from a hardware video port to a DirectDraw surface. This section is a reference to the methods of this interface. For a conceptual overview, see Video-Ports.

The methods of the **IDirectDrawVideoPort** interface can be organized into the following groups:

Color controls	**GetColorControls**
	SetColorControls
Fields and Signals	**GetFieldPolarity**
	GetVideoSignalStatus
Flipping	**Flip**
	SetTargetSurface
Formats	**GetInputFormats**
	GetOutputFormats
Timing and Synchronization	**GetVideoLine**
	WaitForSync
Video control	**StartVideo**
	StopVideo
	UpdateVideo
Zoom factors	**GetBandwidthInfo**

The **IDirectDrawVideoPort** interface, like all COM interfaces, inherits the **IUnknown** interface methods. The **IUnknown** interface supports the following three methods:

AddRef

QueryInterface

Release

You can use the LPDIRECTDRAWVIDEOPORT data type to declare a variable that contains a pointer to an **IDirectDrawVideoPort** interface. The Dvp.h header file declares the LPDIRECTDRAWVIDEOPORT with the following code:

```
typedef struct IDirectDrawVideoPort    FAR *LPDIRECTDRAWVIDEOPORT;
```

IDirectDrawVideoPort::Flip

The **IDirectDrawVideoPort::Flip** method instructs the DirectDrawVideoPort object to write the next frame of video to a new surface.

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

Parameters

lpDDSurface

Address of the DirectDrawSurface object that will receive the next frame of video.

dwFlags

Flip options flags. This parameter can be one of the following values.

DDVPFLIP_VIDEO

The specified surface is to receive the normal video data.

DDVPFLIP_VBI

The specified surface is to receive only the data within the vertical blanking interval.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

Remarks

This method can be used to prevent tearing. Calls to **IDirectDrawVideoPort::Flip** are asynchronous—the actual flip operation will always be synchronized with the vertical blank of the video signal.

IDirectDrawVideoPort::GetBandwidthInfo

The **IDirectDrawVideoPort::GetBandwidthInfo** method retrieves the minimum required overlay zoom factors and device limitations of a video port that uses the provided output pixel format.

```
HRESULT GetBandwidthInfo(
  LPDDPIXELFORMAT lpddpfFormat,
  DWORD dwWidth,
  DWORD dwHeight,
  DWORD dwFlags,
  LPDDVIDEOPORTBANDWIDTH lpBandwidth
);
```

Parameters

lpddpfFormat
Address of a **DDPIXELFORMAT** structure that describes the output pixel format for which bandwidth information will be retrieved.

dwWidth and *dwHeight*
Dimensions of an overlay or video data. These interpretation of these parameters depends on the value specified in the *dwFlags* parameter.

dwFlags
Flags indicating how the method is to interpret the *dwWidth* and *dwHeight* parameters. This parameter can be one of the following values.

DDVPB_OVERLAY
The *dwWidth* and *dwHeight* parameters indicate the size of the source overlay surface. Use this flag when the video port is dependent on the overlay source size.

DDVPB_TYPE
The *dwWidth* and *dwHeight* parameters are not set. The method will retrieve the device's dependency type in the **dwCaps** member of the associated **DDVIDEOPORTBANDWIDTH** structure. Use this flag when you call this method the first time.

DDVPB_VIDEOPORT
The *dwWidth* and *dwHeight* parameters indicate the prescale size of the of the video data that the video port writes to the frame buffer. Use this flag when the video port is dependent on the overlay zoom factor.

lpBandwidth
Address of a **DDVIDEOPORTBANDWIDTH** structure that will be filled with the retrieved bandwidth and device dependency information.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

Remarks

This method will usually be called twice. When you make the first call, specify the DDVPB_TYPE flag in the *dwFlags* parameter to retrieve information about device's overlay dependency type. Subsequent calls using the DDVPB_VIDEOPORT or DDVPB_OVERLAY flags must be interpreted considering the device's dependency type.

IDirectDrawVideoPort::GetColorControls

The **IDirectDrawVideoPort::GetColorControls** method returns the current color control settings associated with the video port.

```
HRESULT GetColorControls(
  LPDDCOLORCONTROL lpColorControl
);
```

Parameters

lpColorControl
Address of a **DDCOLORCONTROL** structure that will be filled with the current settings of the video port's color control.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

Remarks

The **dwFlags** member of the **DDCOLORCONTROL** structure indicate which of the color control options are supported.

IDirectDrawVideoPort::GetInputFormats

The **IDirectDrawVideoPort::GetInputFormat** method retrieves the input formats supported by the DirectDrawVideoPort object.

```
HRESULT GetInputFormats(
  LPDWORD lpNumFormats,
  LPDDPIXELFORMAT lpFormats,
  DWORD dwFlags
);
```

Parameters

lpNumFormats

Address of a variable containing the number of entries that the array at *lpFromats* can hold. If this number is less than the total number of codes, the method fills the array with as many codes as will fit, sets the value at *lpNumFormats* to indicate the total number of codes, and returns DDERR_MOREDATA.

lpFormats

Address of an array of **DDPIXELFORMAT** structures that will be filled in with the input formats supported by this DirectDrawVideoPort object. If this parameter is NULL, the method sets *lpNumFormats* to the number of supported formats and the returns DD_OK.

dwFlags

Flags specifying the part of the video signal for which formats will be enumerated. This parameter can be one of the following values.

DDVPFORMAT_VIDEO

Returns formats for the video data.

DDVPFORMAT_VBI

Returns formats for the VBI data.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_MOREDATA

Remarks

This method can also be used to return the number of formats supported. To do this, set the *lpFormats* parameter to NULL. When the method returns, the variable at *lpNumFormats* contains the total number of supported input formats.

IDirectDrawVideoPort::GetOutputFormats

The **IDirectDrawVideoPort::GetOutputFormats** method retrieves a list of output formats that the DirectDrawVideoPort object supports for a specified input format.

```
HRESULT GetOutputFormats(
  LPDDPIXELFORMAT lpInputFormat,
  LPDWORD lpNumFormats,
  LPDDPIXELFORMAT lpFormats,
  DWORD dwFlags
);
```

Parameters

lpInputFormat
Address of a **DDPIXELFORMAT** structure that describes the input format for which conversion information is requested.

lpNumFormats
Address of a variable containing the number of entries that the array at *lpFromats* can hold. If this number is less than the total number of codes, the method fills the array with as many codes as will fit, sets the value at *lpNumFormats* to indicate the total number of codes, and returns DDERR_MOREDATA.

lpFormats
Address of an array of **DDPIXELFORMAT** structures that will be filled in with the output formats supported by this DirectDrawVideoPort object. If this parameter is NULL, the method sets *lpNumFormats* to the number of supported formats and the returns DD_OK.

dwFlags
Flags specifying the part of the video signal for which formats will be enumerated. This parameter can be one of the following values.

DDVPFORMAT_VIDEO
Returns formats for the video data.

DDVPFORMAT_VBI
Returns formats for the VBI data.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_MOREDATA

IDirectDrawVideoPort::GetFieldPolarity

The **IDirectDrawVideoPort::GetFieldPolarity** method retrieves the status of the video field.

```
HRESULT GetVideoField(
  LPBOOL lpbVideoField
);
```

Parameters

lpbVideoField
Address of a variable that will be set to indicate the current field polarity. This value is set to true if the current video field is the even field of an interlaced video signal and false otherwise.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED
DDERR_VIDEONOTACTIVE

IDirectDrawVideoPort::GetVideoLine

The **IDirectDrawVideoPort::GetVideoLine** method retrieves the current line of video being written to the frame buffer.

```
HRESULT GetVideoLine(
  LPDWORD lpdwLine
);
```

Parameters

lpdwLine

Address of a variable that will be filled with a value indicating the video line currently being written to the frame buffer.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT

DDERR_INVALIDPARAMS

DDERR_UNSUPPORTED

DDERR_VERTICALBLANKINPROGRESS

DDERR_VIDEONOTACTIVE

Remarks

The value this method retrieves reflects the true video line being written, relative to the field height, before any prescaling occurs.

IDirectDrawVideoPort::GetVideoSignalStatus

The **IDirectDrawVideoPort::GetVideoSignalStatus** method retrieves the status of the video signal currently being presented to the video port.

```
HRESULT GetVideoSignalStatus(
  LPDWORD lpdwStatus
);
```

Parameters

lpdwStatus

Address of a variable that will contain a return code indicating the quality of the video signal at the video port. The value will be set to one of the following codes.

DDVPSQ_NOSIGNAL

No video signal is present at the video port.

DDVPSQ_SIGNALOK

A valid video signal is present at the video port.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

IDirectDrawVideoPort::SetColorControls

The **IDirectDrawVideoPort::SetColorControls** method sets the color control settings associated with the video port.

```
HRESULT SetColorControls(
  LPDDCOLORCONTROL lpColorControl
);
```

Parameters

lpColorControl
Address of a **DDCOLORCONTROL** structure containing the new color control settings that will be applied to the video port.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED

IDirectDrawVideoPort::SetTargetSurface

The **IDirectDrawVideoPort::SetTargetSurface** method sets the DirectDraw surface object that will receive the stream of live video data and/or the vertical blank interval data.

```
HRESULT SetTargetSurface(
  LPDIRECTDRAWSURFACE lpDDSurface,
  DWORD dwFlags
);
```

Parameters

lpDDSurface
Address of the DirectDrawSurface object that will receive the video data.

dwFlags
Value specifying the type of target surface.

DDVPTARGET_VIDEO	The specified surface should receive the normal video data and vertical interval data unless a separate surface was attached for this purpose.
DDVPTARGET_VBI	The specified surface should receive the data within the vertical blanking interval.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirectDrawVideoPort::StartVideo, **IDirectDrawVideoPort::StopVideo**, **IDirectDrawVideoPort::UpdateVideo**

IDirectDrawVideoPort::StartVideo

The **IDirectDrawVideoPort::StartVideo** method enables the hardware video port and starts the flow of video data into the currently specified surface.

```
HRESULT StartVideo(
  LPDDVIDEOPORTINFO lpVideoInfo
);
```

Parameters

lpVideoInfo
Address of a pointer to a **DDVIDEOPORTINFO** structure.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_SURFACELOST

See Also

IDirectDrawVideoPort::SetTargetSurface, **IDirectDrawVideoPort::StopVideo**, **IDirectDrawVideoPort::UpdateVideo**

IDirectDrawVideoPort::StopVideo

The **IDirectDrawVideoPort::StopVideo** method stops the flow of video port data into the frame buffer.

```
HRESULT StopVideo();
```

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value is DDERR_INVALIDOBJECT.

See Also

IDirectDrawVideoPort::SetTargetSurface, I**IDirectDrawVideoPort::StartVideo**, **IDirectDrawVideoPort::UpdateVideo**

IDirectDrawVideoPort::UpdateVideo

The **IDirectDrawVideoPort::UpdateVideo** method updates parameters that govern the flow of video data from the video-port to the DirectDrawSurface object.

```
HRESULT UpdateVideo(
  LPDDVIDEOPORTINFO lpVideoInfo
);
```

Parameters

lpVideoInfo
Address of a **DDVIDEOPORTINFO** structure that describes the video transfer parameters.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS

See Also

IDirectDrawVideoPort::SetTargetSurface, **IDirectDrawVideoPort::StartVideo**, **IDirectDrawVideoPort::StopVideo**

IDirectDrawVideoPort::WaitForSync

The **IDirectDrawVideoPort::WaitForSync** method waits for VSYNC or until a given scan line is being drawn.

```
HRESULT WaitForSync(
  DWORD dwFlags,
  DWORD dwLine,
  DWORD dwTimeout
);
```

Parameters

dwFlags

Flag specifying how the method will wait for the video VSYNC or the specified line number.

DDVPWAIT_BEGINVBLANK	Return at the start of the vertical blanking interval.
DDVPWAIT_ENDVBLANK	Return at the end of the vertical blanking interval .
DDVPWAIT_LINE	Return when the video counter either reaches or passes the line specified by the *dwLine* parameter.

dwLine

The video line determining when the method should return, relative to the field height, before prescaling. This parameter is ignored if the *dwFlags* parameter is set to DDVPWAIT_BEGINVBLANK or DDVPWAIT_ENDVBLANK.

dwTimeout

Amount of time, in milliseconds, that the method will wait for the next video vertical blank before timing out. If this parameter is 0, the method waits 3 times the value specified in the **dwMicrosecondsPerField** member of the **DDVIDEOPORTDESC**.

Return Values

If the method succeeds, the return value is DD_OK.

If the method fails, the return value may be one of the following error values:

DDERR_INVALIDOBJECT
DDERR_INVALIDPARAMS
DDERR_UNSUPPORTED
DDERR_VIDEONOTACTIVE
DDERR_WASSTILLDRAWING

Remarks

This method helps the caller synchronize with the video vertical blank interval or with an arbitrary line of video data. The method blocks the calling thread until either the video VSYNC occurs or when the video line counter matches the specified line number.

Functions

This section contains information about the following DirectDraw global functions:

- **DirectDrawCreate**
- **DirectDrawCreateClipper**
- **DirectDrawEnumerate**

DirectDrawCreate

The **DirectDrawCreate** function creates an instance of a DirectDraw object.

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

Parameters

lpGUID

Address of the globally unique identifier (GUID) that represents the driver to be created. This can be NULL to indicate the active display driver, or you can pass one of the following flags to restrict the active display driver's behavior for debugging purposes:

DDCREATE_EMULATIONONLY

The DirectDraw object will use emulation for all features; it will not take advantage of any hardware supported features.

DDCREATE_HARDWAREONLY

The DirectDraw object will never emulate features not supported by the hardware. Attempts to call methods that require unsupported features will fail, returning DDERR_UNSUPPORTED.

lplpDD

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

pUnkOuter

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

Return Values

If the function succeeds, the return value is DD_OK.

If the function fails, the return value may be one of the following error values:

DDERR_DIRECTDRAWALREADYCREATED

DDERR_GENERIC

DDERR_INVALIDDIRECTDRAWGUID

DDERR_INVALIDPARAMS

DDERR_NODIRECTDRAWHW

DDERR_OUTOFMEMORY

Remarks

This function attempts to initialize a DirectDraw object, and it then sets a pointer to the object if the call is successful.

On systems with multiple monitors, specifying NULL for *lpGUID* causes the DirectDraw object to run in emulation mode when the normal cooperative level is set. To make use of hardware acceleration on these systems, you must specify the device's GUID. For more information, see Working with Multiple Monitors.

DirectDrawCreateClipper

The **DirectDrawCreateClipper** function creates an instance of a DirectDrawClipper object not associated with a DirectDraw object.

HRESULT DirectDrawCreateClipper(
DWORD *dwFlags*,
LPDIRECTDRAWCLIPPER FAR **lplpDDClipper*,

IUnknown FAR **pUnkOuter*
);

Parameters

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

lplpDDClipper
Address of a pointer that will be filled with the address of the new DirectDrawClipper object.

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

Return Values

If the function succeeds, the return value is DD_OK.

If the function fails, the return value may be one of the following error values:

DDERR_INVALIDPARAMS
DDERR_OUTOFMEMORY

Remarks

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

To create a DirectDrawClipper object owned by a specific DirectDraw object, use the **IDirectDraw2::CreateClipper** method.

See Also

IDirectDraw2::CreateClipper

DirectDrawEnumerate

The **DirectDrawEnumerate** function enumerates the DirectDraw objects installed on the system. The NULL GUID entry always identifies the primary display device shared with GDI.

HRESULT DirectDrawEnumerate(
LPDDENUMCALLBACK *lpCallback*,

```
  LPVOID lpContext
);
```

Parameters

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

lpContext
Address of an application-defined context that will be passed to the enumeration callback function each time it is called.

Return Values

If the function succeeds, the return value is DD_OK.

If the function fails, the return value is DDERR_INVALIDPARAMS.

Remarks

On systems with multiple monitors, this method enumerates multiple display devices. For more information, see Working with Multiple Monitors.

Callback Functions

This section contains information about the following callback functions used with DirectDraw:

- **DDEnumCallback**
- **EnumModesCallback**
- **EnumSurfacesCallback**
- **EnumVideoCallback**

DDEnumCallback

The **DDEnumCallback** is an application-defined callback function for the **DirectDrawEnumerate** function.

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

Parameters

lpGUID
Address of the unique identifier of the DirectDraw object.

lpDriverDescription
Address of a string containing the driver description.

lpDriverName
Address of a string containing the driver name.

lpContext
Address of an application-defined structure that will be passed to the callback function each time the function is called.

Return Values

The callback function returns DDENUMRET_OK to continue the enumeration.

The callback function returns DDENUMRET_CANCEL to stop it.

Remarks

You can use the LPDDENUMCALLBACK data type to declare a variable that can contain a pointer to this callback function.

EnumModesCallback

The **EnumModesCallback** is an application-defined callback function for the **IDirectDraw2::EnumDisplayModes** method.

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

Parameters

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

lpContext
Address of an application-defined structure that will be passed to the callback function each time the function is called.

Return Values

The callback function returns DDENUMRET_OK to continue the enumeration.

The callback function returns DDENUMRET_CANCEL to stop it.

Remarks

You can use the LPDDENUMMODESCALLBACK data type to declare a variable that can contain a pointer to this callback function.

EnumSurfacesCallback

The **EnumSurfacesCallback** is an application-defined callback function for the **IDirectDrawSurface3::EnumAttachedSurfaces** and **IDirectDrawSurface3::EnumOverlayZOrders** methods.

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

Parameters

lpDDSurface
Address of the surface attached to this surface.

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

lpContext
Address of an application-defined structure that will be passed to the callback function each time the function is called.

Return Values

The callback function returns DDENUMRET_OK to continue the enumeration.

The callback function returns DDENUMRET_CANCEL to stop it.

Remarks

You can use the LPDDENUMSURFACESCALLBACK data type to declare a variable that can contain a pointer to this callback function.

EnumVideoCallback

The **EnumVideoCallback** is an application-defined callback procedure for the **IDDVideoPortContainer::EnumVideoPorts** method.

```
HRESULT WINAPI EnumVideoCallback(
```

```
  LPDDVIDEOPORTCAPS lpDDVideoPortCaps,
  LPVOID lpContext
);
```

Parameters

lpDDVideoPortCaps
Pointer to the **DDVIDEOPORTCAPS** structure that contains the video port information, including the ID and capabilities. This data is read-only.

lpContext
Pointer to a caller-defined structure that is passed to the member every time it is called.

Return Values

The callback function returns DDENUMRET_OK to continue the enumeration.

The callback function returns DDENUMRET_CANCEL to stop it.

Remarks

Video-port related functions cannot be called from inside the **EnumVideoCallback** function. Attempts to do so will fail, returning DDERR_CURRENTLYNOTAVAIL.

You can use the LPDDENUMVIDEOCALLBACK data type to declare a variable that can contain a pointer to this callback function.

Structures

This section contains information about the following structures used with DirectDraw:

- **DDBLTBATCH**
- **DDBLTFX**
- **DDCAPS**
- **DDCOLORCONTROL**
- **DDCOLORKEY**
- **DDOVERLAYFX**
- **DDPIXELFORMAT**
- **DDSCAPS**
- **DDSURFACEDESC**
- **DDVIDEOPORTBANDWIDTH**
- **DDVIDEOPORTCAPS**
- **DDVIDEOPORTCONNECT**

- **DDVIDEOPORTDESC**
- **DDVIDEOPORTINFO**
- **DDVIDEOPORTSTATUS**

DDBLTBATCH

The **DDBLTBATCH** structure passes blit operations to the **IDirectDrawSurface3::BltBatch** method.

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

Members

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

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

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

dwFlags
Optional control flags.

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

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

DDBLT_ALPHADESTNEG
Indicates that the destination surface becomes more transparent as the alpha value increases (0 is opaque).

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

DDBLT_ALPHAEDGEBLEND
Uses the **dwAlphaEdgeBlend** member of the **DDBLTFX** structure as

the alpha channel for the edges of the image that border the color key colors.

DDBLT_ALPHASRC

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

DDBLT_ALPHASRCCONSTOVERRIDE

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

DDBLT_ALPHASRCNEG

Indicates that the source surface becomes more transparent as the alpha value increases (0 is opaque).

DDBLT_ALPHASRCSURFACEOVERRIDE

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

DDBLT_ASYNC

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

DDBLT_COLORFILL

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

DDBLT_DDFX

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

DDBLT_DDROPS

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

DDBLT_KEYDEST

Uses the color key associated with the destination surface.

DDBLT_KEYDESTOVERRIDE

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

DDBLT_KEYSRC

Uses the color key associated with the source surface.

DDBLT_KEYSRCOVERRIDE

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

DDBLT_ROP

Uses the **dwROP** member of the **DDBLTFX** structure for the ROP for this blit. The ROPs are the same as those defined in the Win32 API.

DDBLT_ROTATIONANGLE

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

DDBLT_ZBUFFER

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

DDBLT_ZBUFFERDESTCONSTOVERRIDE

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

DDBLT_ZBUFFERDESTOVERRIDE

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

DDBLT_ZBUFFERSRCCONSTOVERRIDE

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

DDBLT_ZBUFFERSRCOVERRIDE

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

lpDDBltFx

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

DDBLTFX

The **DDBLTFX** structure passes raster operations, effects, and override information to the **IDirectDrawSurface3::Blt** method. This structure is also part of the **DDBLTBATCH** structure used with the **IDirectDrawSurface3::BltBatch** method.

```
typedef struct _DDBLTFX{
   DWORD dwSize;
   DWORD dwDDFX;
   DWORD dwROP;
   DWORD dwDDROP;
   DWORD dwRotationAngle;
   DWORD dwZBufferOpCode;
   DWORD dwZBufferLow;
   DWORD dwZBufferHigh;
   DWORD dwZBufferBaseDest;
   DWORD dwZDestConstBitDepth;
union
```

```
{
    DWORD           dwZDestConst;
    LPDIRECTDRAWSURFACE lpDDSZBufferDest;
};
  DWORD dwZSrcConstBitDepth;
union
{
    DWORD           dwZSrcConst;
    LPDIRECTDRAWSURFACE lpDDSZBufferSrc;

};
  DWORD dwAlphaEdgeBlendBitDepth;
  DWORD dwAlphaEdgeBlend;
  DWORD dwReserved;
  DWORD dwAlphaDestConstBitDepth;
union
{
    DWORD           dwAlphaDestConst;
    LPDIRECTDRAWSURFACE lpDDSAlphaDest;
};
  DWORD dwAlphaSrcConstBitDepth;
union
{
    DWORD           dwAlphaSrcConst;
    LPDIRECTDRAWSURFACE lpDDSAlphaSrc;
};
union
{
    DWORD           dwFillColor;
    DWORD           dwFillDepth;
    DWORD           dwFillPixel;
    LPDIRECTDRAWSURFACE lpDDSPattern;
};
DDCOLORKEY ddckDestColorkey;
DDCOLORKEY ddckSrcColorkey;
} DDBLTFX,FAR* LPDDBLTFX;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwDDFX

Type of FX operations.

DDBLTFX_ARITHSTRETCHY

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

DDBLTFX_MIRRORLEFTRIGHT

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

DDBLTFX_MIRRORUPDOWN

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

DDBLTFX_NOTEARING

Schedules this blit to avoid tearing.

DDBLTFX_ROTATE180

Rotates the surface 180 degrees clockwise during this blit.

DDBLTFX_ROTATE270

Rotates the surface 270 degrees clockwise during this blit.

DDBLTFX_ROTATE90

Rotates the surface 90 degrees clockwise during this blit.

DDBLTFX_ZBUFFERBASEDEST

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

DDBLTFX_ZBUFFERRANGE

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

dwROP

Win32 raster operations. You can retrieve a list of supported raster operations by calling the **IDirectDraw2::GetCaps** method.

dwDDROP

DirectDraw raster operations.

dwRotationAngle

Rotation angle for the blit.

dwZBufferOpCode

Z-buffer compares.

dwZBufferLow

Low limit of a z-buffer.

dwZBufferHigh

High limit of a z-buffer.

dwZBufferBaseDest

Destination base value of a z-buffer.

dwZDestConstBitDepth

Bit depth of the destination z-constant.

dwZDestConst

Constant used as the z-buffer destination.

lpDDSZBufferDest
Surface used as the z-buffer destination.

dwZSrcConstBitDepth
Bit depth of the source z-constant.

dwZSrcConst
Constant used as the z-buffer source.

lpDDSZBufferSrc
Surface used as the z-buffer source.

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

dwAlphaEdgeBlend
Alpha constant used for edge blending.

dwReserved
Reserved for future use.

dwAlphaDestConstBitDepth
Bit depth of the destination alpha constant.

dwAlphaDestConst
Constant used as the alpha channel destination.

lpDDSAlphaDest
Surface used as the alpha channel destination.

dwAlphaSrcConstBitDepth
Bit depth of the source alpha constant.

dwAlphaSrcConst
Constant used as the alpha channel source.

lpDDSAlphaSrc
Surface used as the alpha channel source.

dwFillColor
Color used to fill a surface when DDBLT_COLORFILL is specified. This value must be a pixel appropriate to the pixel format of the destination surface. For a palettized surface it would be a palette index, and for a 16-bit RGB surface it would be a 16-bit pixel value.

dwFillDepth
Depth value for the z-buffer.

dwFillPixel
Pixel value for RGBA or RGBZ fills. Applications that use RGBZ fills should use this member, not **dwFillColor** or **dwFillDepth**.

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

ddckDestColorkey
Destination color key override.

ddckSrcColorkey

Source color key override.

DDCAPS

The **DDCAPS** structure represents the capabilities of the hardware exposed through the DirectDraw object. This structure contains a **DDSCAPS** structure used in this context to describe what kinds of DirectDrawSurface objects can be created. It may not be possible to simultaneously create all of the surfaces described by these capabilities. This structure is used with the **IDirectDraw2::GetCaps** method.

```
typedef struct _DDCAPS {
    DWORD   dwSize;
    DWORD   dwCaps;               // driver-specific caps
    DWORD   dwCaps2;              // more driver-specific caps
    DWORD   dwCKeyCaps;           // color key caps
    DWORD   dwFXCaps;             // stretching and effects caps
    DWORD   dwFXAlphaCaps;        // alpha caps
    DWORD   dwPalCaps;            // palette caps
    DWORD   dwSVCaps;             // stereo vision caps
    DWORD   dwAlphaBltConstBitDepths;      // alpha bit-depth members
    DWORD   dwAlphaBltPixelBitDepths;      //  .
    DWORD   dwAlphaBltSurfaceBitDepths;    //  .
    DWORD   dwAlphaOverlayConstBitDepths;  //  .
    DWORD   dwAlphaOverlayPixelBitDepths;  //  .
    DWORD   dwAlphaOverlaySurfaceBitDepths; //  .
    DWORD   dwZBufferBitDepths;     // Z-buffer bit depth
    DWORD   dwVidMemTotal;          // total video memory
    DWORD   dwVidMemFree;           // total free video memory
    DWORD   dwMaxVisibleOverlays;   // maximum visible overlays
    DWORD   dwCurrVisibleOverlays;  // overlays currently visible
    DWORD   dwNumFourCCCodes;       // number of supported FOURCC codes
    DWORD   dwAlignBoundarySrc;     // overlay alignment restrictions
    DWORD   dwAlignSizeSrc;         //  .
    DWORD   dwAlignBoundaryDest;    //  .
    DWORD   dwAlignSizeDest;        //  .
    DWORD   dwAlignStrideAlign;     // stride alignment
    DWORD   dwRops[DD_ROP_SPACE];   // supported raster ops
    DDSCAPS  ddsCaps;               // general surface caps
    DWORD   dwMinOverlayStretch;    // overlay stretch factors
    DWORD   dwMaxOverlayStretch;    //  .
    DWORD   dwMinLiveVideoStretch;  // obsolete
    DWORD   dwMaxLiveVideoStretch;  //  .
    DWORD   dwMinHwCodecStretch;    //  .
    DWORD   dwMaxHwCodecStretch;    //  .
    DWORD   dwReserved1;            // reserved
    DWORD   dwReserved2;            //  .
```

```
    DWORD   dwReserved3;        //  .
    DWORD   dwSVBCaps;          // system-to-video blit related caps
    DWORD   dwSVBCKeyCaps;      //  .
    DWORD   dwSVBFXCaps;        //  .
    DWORD   dwSVBRops[DD_ROP_SPACE]; //  .
    DWORD   dwSVBCaps;          // video-to-system blit related caps
    DWORD   dwVSBCKeyCaps;      //  .
    DWORD   dwVSBFXCaps;        //  .
    DWORD   dwVSBRops[DD_ROP_SPACE]; //  .
    DWORD   dwSSBCaps;          // system-to-system blit related caps
    DWORD   dwSSBCKeyCaps;      //  .
    DWORD   dwSSBCFXCaps;       //  .
    DWORD   dwSSBRops[DD_ROP_SPACE]; //  .
    DWORD   dwMaxVideoPorts;    // maximum number of live video ports
    DWORD   dwCurrVideoPorts;   // current number of live video ports
    DWORD   dwSVBCaps2;         // additional system-to-video blit caps
    DWORD   dwNLVBCaps;         // nonlocal-to-local video memory blit caps
    DWORD   dwNLVBCaps2;        //  .
    DWORD   dwNLVBCKeyCaps;     //  .
    DWORD   dwNLVBFXCaps;       //  .
    DWORD   dwNLVBRops[DD_ROP_SPACE];//  .
    DWORD   dwReserved4;        // reserved
    DWORD   dwReserved5;        //  .
    DWORD   dwReserved6;        //  .
} DDCAPS,FAR* LPDDCAPS;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwCaps

Driver-specific capabilities.

DDCAPS_3D

Indicates that the display hardware has 3-D acceleration.

DDCAPS_ALIGNBOUNDARYDEST

Indicates that DirectDraw will support only those overlay destination rectangles with the x-axis aligned to the **dwAlignBoundaryDest** boundaries of the surface.

DDCAPS_ALIGNBOUNDARYSRC

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

DDCAPS_ALIGNSIZEDEST

Indicates that DirectDraw will support only those overlay destination rectangles whose x-axis sizes, in pixels, are **dwAlignSizeDest** multiples.

DDCAPS_ALIGNSIZESRC

Indicates that DirectDraw will support only those overlay source rectangles whose x-axis sizes, in pixels, are **dwAlignSizeSrc** multiples.

DDCAPS_ALIGNSTRIDE

Indicates that DirectDraw will create display memory surfaces that have a stride alignment equal to the **dwAlignStrideAlign** value.

DDCAPS_ALPHA

Indicates that the display hardware supports an alpha channel during blit operations.

DDCAPS_BANKSWITCHED

Indicates that the display hardware is bank-switched and is potentially very slow at random access to display memory.

DDCAPS_BLT

Indicates that display hardware is capable of blit operations.

DDCAPS_BLTCOLORFILL

Indicates that display hardware is capable of color filling with a blitter.

DDCAPS_BLTDEPTHFILL

Indicates that display hardware is capable of depth filling z-buffers with a blitter.

DDCAPS_BLTFOURCC

Indicates that display hardware is capable of color-space conversions during blit operations.

DDCAPS_BLTQUEUE

Indicates that display hardware is capable of asynchronous blit operations.

DDCAPS_BLTSTRETCH

Indicates that display hardware is capable of stretching during blit operations.

DDCAPS_CANBLTSYSMEM

Indicates that display hardware is capable of blitting to or from system memory.

DDCAPS_CANCLIP

Indicates that display hardware is capable of clipping with blitting.

DDCAPS_CANCLIPSTRETCHED

Indicates that display hardware is capable of clipping while stretch blitting.

DDCAPS_COLORKEY

Supports some form of color key in either overlay or blit operations.

More specific color key capability information can be found in the **dwCKeyCaps** member.

DDCAPS_COLORKEYHWASSIST

Indicates that the color key is partially hardware assisted. This means that other resources (CPU and/or video memory) might be used. If this bit is not set, full hardware support is in place.

DDCAPS_GDI

Indicates that display hardware is shared with GDI.

DDCAPS_NOHARDWARE

Indicates that there is no hardware support.

DDCAPS_OVERLAY

Indicates that display hardware supports overlays.

DDCAPS_OVERLAYCANTCLIP

Indicates that display hardware supports overlays but cannot clip them.

DDCAPS_OVERLAYFOURCC

Indicates that overlay hardware is capable of color-space conversions during overlay operations.

DDCAPS_OVERLAYSTRETCH

Indicates that overlay hardware is capable of stretching. The **dwMinOverlayStretch** and **dwMaxOverlayStretch** members contain valid data.

DDCAPS_PALETTE

Indicates that DirectDraw is capable of creating and supporting DirectDrawPalette objects for more surfaces than only the primary surface.

DDCAPS_PALETTEVSYNC

Indicates that DirectDraw is capable of updating a palette synchronized with the vertical refresh.

DDCAPS_READSCANLINE

Indicates that display hardware is capable of returning the current scan line.

DDCAPS_STEREOVIEW

Indicates that display hardware has stereo vision capabilities.

DDCAPS_VBI

Indicates that display hardware is capable of generating a vertical-blank interrupt.

DDCAPS_ZBLTS

Supports the use of z-buffers with blit operations.

DDCAPS_ZOVERLAYS

Supports the use of the **IDirectDrawSurface3::UpdateOverlayZOrder**

method as a z-value for overlays to control their layering.

dwCaps2

More driver-specific capabilities.

DDCAPS2_AUTOFLIPOVERLAY

The overlay can be automatically flipped to the next surface in the flip chain each time a video port VSYNC occurs, allowing the video port and the overlay to double buffer the video without CPU overhead. This option is only valid when the surface is receiving data from a video port. If the video port data is non-interlaced or non-interleaved, it will flip on every VSYNC. If the data is being interleaved in memory, it will flip on every other VSYNC.

DDCAPS2_CANBOBINTERLEAVED

The overlay hardware can display each field individually of an interlaced video stream while it is interleaved in memory without causing any artifacts that might normally occur without special hardware support. This option is only valid when the surface is receiving data from a video port and is only valid when the video is zoomed at least 2X in the vertical direction.

DDCAPS2_CANBOBNONINTERLEAVED

The overlay hardware can display each field individually of an interlaced video stream while it is not interleaved in memory without causing any artifacts that might normally occur without special hardware support. This option is only valid when the surface is receiving data from a video port and is only valid when the video is zoomed at least 2X in the vertical direction.

DDCAPS2_CANDROPZ16BIT

Sixteen-bit RGBZ values can be converted into sixteen-bit RGB values. (The system does not support eight-bit conversions.)

DDCAPS2_CANFLIPODDEVEN

The driver is capable of performing odd and even flip operations, as specified by the DDFLIP_ODD and DDFLIP_EVEN flags used with the **IDirectDrawSurface3::Flip** method.

DDCAPS2_CANSMOOTHINTERLEAVED

Overlay can display each field of interlaced data individually while it is interleaved in memory without causing jittery artifacts.

DDCAPS2_CANSMOOTHNONINTERLEAVED

Overlay can display each field of interlaced data individually while it is not interleaved in memory without causing jittery artifacts.

DDCAPS2_CERTIFIED

Indicates that display hardware is certified.

DDCAPS2_COLORCONTROLPRIMARY

The primary surface contains color controls (gamma, etc.)

DDCAPS2_COLORCONTROLOVERLAY

The overlay surface contains color controls (brightness, sharpness, etc.)

DDCAPS2_NO2DDURING3DSCENE

Indicates that 2-D operations such as **IDirectDrawSurface3::Blt** and **IDirectDrawSurface3::Lock** cannot be performed on any surfaces that Direct3D® is using between calls to the **IDirect3DDevice2::BeginScene** and **IDirect3DDevice2::EndScene** methods.

DDCAPS2_NONLOCALVIDMEM

Indicates that the display driver supports surfaces in non-local video memory.

DDCAPS2_NONLOCALVIDMEMCAPS

Indicates that blit capabilities for non-local video memory surfaces differ from local video memory surfaces. If this flag is present, the DDCAPS2_NONLOCALVIDMEM flag will also be present.

DDCAPS2_NOPAGELOCKREQUIRED

DMA blit operations are supported on system memory surfaces that are not page locked.

DDCAPS2_VIDEOPORT

Indicates that display hardware supports live video.

DDCAPS2_WIDESURFACES

Indicates that the display surfaces supports surfaces wider than the primary surface.

dwCKeyCaps

Color-key capabilities.

DDCKEYCAPS_DESTBLT

Supports transparent blitting with a color key that identifies the replaceable bits of the destination surface for RGB colors.

DDCKEYCAPS_DESTBLTCLRSPACE

Supports transparent blitting with a color space that identifies the replaceable bits of the destination surface for RGB colors.

DDCKEYCAPS_DESTBLTCLRSPACEYUV

Supports transparent blitting with a color space that identifies the replaceable bits of the destination surface for YUV colors.

DDCKEYCAPS_DESTBLTYUV

Supports transparent blitting with a color key that identifies the replaceable bits of the destination surface for YUV colors.

DDCKEYCAPS_DESTOVERLAY

Supports overlaying with color keying of the replaceable bits of the destination surface being overlaid for RGB colors.

DDCKEYCAPS_DESTOVERLAYCLRSPACE

Supports a color space as the color key for the destination of RGB colors.

DDCKEYCAPS_DESTOVERLAYCLRSPACEYUV

Supports a color space as the color key for the destination of YUV colors.

DDCKEYCAPS_DESTOVERLAYONEACTIVE

Supports only one active destination color key value for visible overlay surfaces .

DDCKEYCAPS_DESTOVERLAYYUV

Supports overlaying using color keying of the replaceable bits of the destination surface being overlaid for YUV colors.

DDCKEYCAPS_NOCOSTOVERLAY

Indicates there are no bandwidth trade-offs for using the color key with an overlay.

DDCKEYCAPS_SRCBLT

Supports transparent blitting using the color key for the source with this surface for RGB colors.

DDCKEYCAPS_SRCBLTCLRSPACE

Supports transparent blitting using a color space for the source with this surface for RGB colors.

DDCKEYCAPS_SRCBLTCLRSPACEYUV

Supports transparent blitting using a color space for the source with this surface for YUV colors.

DDCKEYCAPS_SRCBLTYUV

Supports transparent blitting using the color key for the source with this surface for YUV colors.

DDCKEYCAPS_SRCOVERLAY

Supports overlaying using the color key for the source with this overlay surface for RGB colors.

DDCKEYCAPS_SRCOVERLAYCLRSPACE

Supports overlaying using a color space as the source color key for the overlay surface for RGB colors.

DDCKEYCAPS_SRCOVERLAYCLRSPACEYUV

Supports overlaying using a color space as the source color key for the overlay surface for YUV colors.

DDCKEYCAPS_SRCOVERLAYONEACTIVE

Supports only one active source color key value for visible overlay surfaces.

DDCKEYCAPS_SRCOVERLAYYUV

Supports overlaying using the color key for the source with this overlay surface for YUV colors.

dwFXCaps

Driver-specific stretching and effects capabilities.

DDFXCAPS_BLTARITHSTRETCHY

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during a blit operation. Occurs along the y-axis (vertically).

DDFXCAPS_BLTARITHSTRETCHYN

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during a blit operation. Occurs along the y-axis (vertically), and works only for integer stretching (×1, ×2, and so on).

DDFXCAPS_BLTMIRRORLEFTRIGHT

Supports mirroring left to right in a blit operation.

DDFXCAPS_BLTMIRRORUPDOWN

Supports mirroring top to bottom in a blit operation.

DDFXCAPS_BLTROTATION

Supports arbitrary rotation in a blit operation.

DDFXCAPS_BLTROTATION90

Supports 90-degree rotations in a blit operation.

DDFXCAPS_BLTSHRINKX

Supports arbitrary shrinking of a surface along the x-axis (horizontally). This flag is valid only for blit operations.

DDFXCAPS_BLTSHRINKXN

Supports integer shrinking (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is valid only for blit operations.

DDFXCAPS_BLTSHRINKY

Supports arbitrary shrinking of a surface along the y-axis (vertically). This flag is valid only for blit operations.

DDFXCAPS_BLTSHRINKYN

Supports integer shrinking (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is valid only for blit operations.

DDFXCAPS_BLTSTRETCHX

Supports arbitrary stretching of a surface along the x-axis (horizontally). This flag is valid only for blit operations.

DDFXCAPS_BLTSTRETCHXN

Supports integer stretching (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is valid only for blit operations.

DDFXCAPS_BLTSTRETCHY

Supports arbitrary stretching of a surface along the y-axis (vertically). This flag is valid only for blit operations.

DDFXCAPS_BLTSTRETCHYN

Supports integer stretching (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is valid only for blit operations.

DDFXCAPS_OVERLAYARITHSTRETCHY

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during an overlay operation. Occurs along the y-axis (vertically).

DDFXCAPS_OVERLAYARITHSTRETCHYN

Uses arithmetic operations, rather than pixel-doubling techniques, to stretch and shrink surfaces during an overlay operation. Occurs along the y-axis (vertically), and works only for integer stretching (×1, ×2, and so on).

DDFXCAPS_OVERLAYMIRRORLEFTRIGHT

Supports mirroring of overlays around the vertical axis.

DDFXCAPS_OVERLAYMIRRORUPDOWN

Supports mirroring of overlays across the horizontal axis.

DDFXCAPS_OVERLAYSHRINKX

Supports arbitrary shrinking of a surface along the x-axis (horizontally). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKXN

Supports integer shrinking (×1, ×2, and so on) of a surface along the x-axis (horizontally). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKY

Supports arbitrary shrinking of a surface along the y-axis (vertically). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSHRINKYN

Supports integer shrinking (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that shrinking is available.

DDFXCAPS_OVERLAYSTRETCHX

Supports arbitrary stretching of a surface along the x-axis (horizontally). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHXN

Supports integer stretching (×1, ×2, and so on) of a surface along the x-

axis (horizontally). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHY

Supports arbitrary stretching of a surface along the y-axis (vertically). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that stretching is available.

DDFXCAPS_OVERLAYSTRETCHYN

Supports integer stretching (×1, ×2, and so on) of a surface along the y-axis (vertically). This flag is valid only for DDSCAPS_OVERLAY surfaces. This flag indicates only the capabilities of a surface; it does not indicate that stretching is available.

dwFXAlphaCaps

Driver-specific alpha capabilities.

DDFXALPHACAPS_BLTALPHAEDGEBLEND

Supports alpha blending around the edge of a source color-keyed surface. Used for blit operations.

DDFXALPHACAPS_BLTALPHAPIXELS

Supports alpha information in pixel format. The bit depth of alpha information in the pixel format can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for blit operations.

DDFXALPHACAPS_BLTALPHAPIXELSNEG

Supports alpha information in pixel format. The bit depth of alpha information in the pixel format can be 1, 2, 4, or 8. The alpha value becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can be used only if DDCAPS_ALPHA is set. Used for blit operations.

DDFXALPHACAPS_BLTALPHASURFACES

Supports alpha-only surfaces. The bit depth of an alpha-only surface can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for blit operations.

DDFXALPHACAPS_BLTALPHASURFACESNEG

Indicates that the alpha channel becomes more transparent as the alpha value increases. The depth of the alpha channel data can be 1, 2, 4, or 8. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can be set only if DDCAPS_ALPHA has been set. Used for blit operations.

DDFXALPHACAPS_OVERLAYALPHAEDGEBLEND

Supports alpha blending around the edge of a source color-keyed surface. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHAPIXELS

Supports alpha information in pixel format. The bit depth of alpha information in pixel format can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHAPIXELSNEG

Supports alpha information in pixel format. The bit depth of alpha information in pixel format can be 1, 2, 4, or 8. The alpha value becomes more transparent as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can be used only if DDCAPS_ALPHA has been set. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHASURFACES

Supports alpha-only surfaces. The bit depth of an alpha-only surface can be 1, 2, 4, or 8. The alpha value becomes more opaque as the alpha value increases. Regardless of the depth of the alpha information, 0 is always the fully transparent value. Used for overlays.

DDFXALPHACAPS_OVERLAYALPHASURFACESNEG

Indicates that the alpha channel becomes more transparent as the alpha value increases. The depth of the alpha channel data can be 1, 2, 4, or 8. Regardless of the depth of the alpha information, 0 is always the fully opaque value. This flag can be used only if DDCAPS_ALPHA has been set. Used for overlays.

dwPalCaps

Palette capabilities.

DDPCAPS_1BIT

Indicates that the index is 1 bit. There are two entries in the color table.

DDPCAPS_2BIT

Indicates that the index is 2 bits. There are four entries in the color table.

DDPCAPS_4BIT

Indicates that the index is 4 bits. There are 16 entries in the color table.

DDPCAPS_8BIT

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

DDPCAPS_8BITENTRIES

Specifies an index to an 8-bit color index. This field is valid only when used with the DDPCAPS_1BIT, DDPCAPS_2BIT, or DDPCAPS_4BIT capability and when the target surface is in 8 bits per pixel (bpp). Each color entry is 1 byte long and is an index to an 8-bpp palette on the

destination surface.

DDPCAPS_ALLOW256

Indicates that this palette can have all 256 entries defined.

DDPCAPS_PRIMARYSURFACE

Indicates that the palette is attached to the primary surface. Changing the palette has an immediate effect on the display unless the DDPCAPS_VSYNC capability is specified and supported.

DDPCAPS_PRIMARYSURFACELEFT

Indicates that the palette is attached to the primary surface on the left. Changing the palette has an immediate effect on the display unless the DDPCAPS_VSYNC capability is specified and supported.

DDPCAPS_VSYNC

Indicates that the palette can be modified synchronously with the monitor's refresh rate.

dwSVCaps

Stereo vision capabilities.

DDSVCAPS_ENIGMA

Indicates that the stereo view is accomplished using Enigma encoding.

DDSVCAPS_FLICKER

Indicates that the stereo view is accomplished using high-frequency flickering.

DDSVCAPS_REDBLUE

Indicates that the stereo view is accomplished when the viewer looks at the image through red and blue filters placed over the left and right eyes. All images must adapt their color spaces for this process.

DDSVCAPS_SPLIT

Indicates that the stereo view is accomplished with split-screen technology.

dwAlphaBltConstBitDepths

DDBD_2, DDBD_4, or DDBD_8. (Indicates 2-, 4-, or 8-bits per pixel.)

dwAlphaBltPixelBitDepths

DDBD_1, DDBD_2, DDBD_4, or DDBD_8. (Indicates 1-, 2-, 4-, or 8-bits per pixel.)

dwAlphaBltSurfaceBitDepths

DDBD_1, DDBD_2, DDBD_4, or DDBD_8. (Indicates 1-, 2-, 4-, or 8-bits per pixel.)

dwAlphaOverlayConstBitDepths

DDBD_2, DDBD_4, or DDBD_8. (Indicates 2-, 4-, or 8-bits per pixel.)

dwAlphaOverlayPixelBitDepths

DDBD_1, DDBD_2, DDBD_4, or DDBD_8. (Indicates 1-, 2-, 4-, or 8-bits per pixel.)

dwAlphaOverlaySurfaceBitDepths
DDBD_1, DDBD_2, DDBD_4, or DDBD_8. (Indicates 1-, 2-, 4-, or 8-bits per pixel.)

dwZBufferBitDepths
DDBD_8, DDBD_16, or DDBD_24. (Indicates 8-, 16-, 24-bits per pixel.) 32-bit z-buffers are not supported.

dwVidMemTotal
Total amount of display memory.

dwVidMemFree
Amount of free display memory.

dwMaxVisibleOverlays
Maximum number of visible overlays.

dwCurrVisibleOverlays
Current number of visible overlays.

dwNumFourCCCodes
Number of FourCC codes.

dwAlignBoundarySrc
Source rectangle alignment for an overlay surface, in pixels.

dwAlignSizeSrc
Source rectangle size alignment for an overlay surface, in pixels. Overlay source rectangles must have a pixel width that is a multiple of this value.

dwAlignBoundaryDest
Destination rectangle alignment for an overlay surface, in pixels.

dwAlignSizeDest
Destination rectangle size alignment for an overlay surface, in pixels. Overlay destination rectangles must have a pixel width that is a multiple of this value.

dwAlignStrideAlign
Stride alignment.

dwRops[DD_ROP_SPACE]
Raster operations supported.

ddsCaps
DDSCAPS structure with general capabilities.

dwMinOverlayStretch and **dwMaxOverlayStretch**
Minimum and maximum overlay stretch factors multiplied by 1000. For example, 1.3 = 1300.

dwMinLiveVideoStretch and dwMaxLiveVideoStretch
These members are obsolete; do not use.

dwMinHwCodecStretch and **dwMaxHwCodecStretch**
These members are obsolete; do not use.

dwReserved1, **dwReserved2**, and **dwReserved3**
Reserved for future use.

dwSVBCaps
Driver-specific capabilities for system-memory-to-display-memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps** member.

dwSVBCKeyCaps
Driver color-key capabilities for system-memory-to-display-memory blits. Valid flags are identical to the blit-related flags used with for the **dwCKeyCaps** member.

dwSVBFXCaps
Driver FX capabilities for system-memory-to-display-memory blits. Valid flags are identical to the blit-related flags used with the **dwFXCaps** member.

dwSVBRops[DD_ROP_SPACE]
Raster operations supported for system-memory-to-display-memory blits.

dwVSBCaps
Driver-specific capabilities for display-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps** member.

dwVSBCKeyCaps
Driver color-key capabilities for display-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with for the **dwCKeyCaps** member.

dwVSBFXCaps
Driver FX capabilities for display-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with the **dwFXCaps** member.

dwVSBRops[DD_ROP_SPACE]
Raster operations supported for display-memory-to-system-memory blits.

dwSSBCaps
Driver-specific capabilities for system-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps** member.

dwSSBCKeyCaps
Driver color-key capabilities for system-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with for the **dwCKeyCaps** member.

dwSSBCFXCaps
Driver FX capabilities for system-memory-to-system-memory blits. Valid flags are identical to the blit-related flags used with the **dwFXCaps** member.

dwSSBRops[DD_ROP_SPACE]
Raster operations supported for system-memory-to-system-memory blits.

dwMaxVideoPorts
Maximum number of live video ports.

dwCurrVideoPorts
Current number of live video ports.

dwSVBCaps2
More driver-specific capabilities for system-memory-to-video-memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps2** member.

dwNLVBCaps

Driver-specific capabilities for nonlocal-to-local video memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps** member.

dwNLVBCaps2
More driver-specific capabilities for nonlocal-to-local video memory blits. Valid flags are identical to the blit-related flags used with the **dwCaps2** member.

dwNLVBCKeyCaps
Driver color-key capabilities for nonlocal-to-local video memory blits. Valid flags are identical to the blit-related flags used with for the **dwCKeyCaps** member.

dwNLVBFXCaps
Driver FX capabilities for nonlocal-to-local video memory blits. Valid flags are identical to the blit-related flags used with the **dwFXCaps** member.

dwNLVBRops[DD_ROP_SPACE]
Raster operations supported for nonlocal-to-local video memory blits.

dwReserved4, dwReserved5, and dwReserved6
Reserved for future use.

DDCOLORCONTROL

The **DDCOLORCONTROL** structure defines the color controls associated with a DirectDrawVideoPortObject, an overlay surface, or a primary surface.

```
typedef struct _DDCOLORCONTROL {
    DWORD   dwSize;
    DWORD   dwFlags;
    LONG    lBrightness;
    LONG    lContrast;
    LONG    lHue;
    LONG    lSaturation;
    LONG    lSharpness;
    LONG    lGamma;
    LONG    lColorEnable;
    DWORD   dwReserved1;
} DDCOLORCONTROL, FAR *LPDDCOLORCONTROL;
```

Members

dwSize
The the size of the structure, in bytes. This member must be initialized before use.

dwFlags
Flags specifying which structure members contain valid data . When the structure is returned by the **IDirectDrawColorControl::GetColorControls** method, it also indicates which options are supported by the device.

DDCOLOR_BRIGHTNESS	The **iBrightness** member contains valid data.
DDCOLOR_CONTRAST	The **iContrast** member contains valid data.
DDCOLOR_COLORENABLE	The **lColorEnable** member contains valid data.
DDCOLOR_GAMMA	The **iGamma** member contains valid data.
DDCOLOR_HUE	The **iHue** member contains valid data.
DDCOLOR_SATURATION	The **iSaturation** member contains valid data.
DDCOLOR_SHARPNESS	The **iSharpness** member contains valid data.

lBrightness
Luminance intensity (Black Level) in IRE units*100. Range is 0 to 10,000. The default is 750 (7.5 IRE)

lContrast
Relative difference between higher and lower intensity luminance values in IRE units*100. The valid range is 0 to 20,000. The default value is 10,000 (100 IRE). Higher values of contrast cause darker luminance values to tend towards black, and cause lighter luminance values to tend towards white. Lower values of contrast cause all luminance values to move towards the middle luminance values.

lHue
Phase relationship of the chrominance components. Hue is specified in degrees and the valid range is -180 to 180. The default is 0.

lSaturation
Color intensity in IRE units*100. The valid range is 0 to 20,000. The default value is 10,000 (100 IRE).

lSharpness
Sharpness in arbitrary units. The valid range is 0 to 10. The default value is 5.

lGamma
Controls the amount of gamma correction applied to the luminance values. The valid range is 1 to 500 gamma units, with a default of 1.

lColorEnable
Flag indicating whether color is used. If this member is zero, color is not used; if it is 1, then color is used. The default value is 1.

dwReserved1
This member is reserved.

DDCOLORKEY

The **DDCOLORKEY** structure describes a source color key, destination color key, or color space. A color key is specified if the low and high range values are the same. This structure is used with the **IDirectDrawSurface3::GetColorKey** and **IDirectDrawSurface3::SetColorKey** methods.

```
typedef struct _DDCOLORKEY{
```

```
    DWORD dwColorSpaceLowValue;
    DWORD dwColorSpaceHighValue;
} DDCOLORKEY,FAR* LPDDCOLORKEY;
```

Members

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

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

DDOVERLAYFX

The **DDOVERLAYFX** structure passes override information to the **IDirectDrawSurface3::UpdateOverlay** method.

```
typedef struct _DDOVERLAYFX{
    DWORD dwSize;
    DWORD dwAlphaEdgeBlendBitDepth;
    DWORD dwAlphaEdgeBlend;
    DWORD dwReserved;
    DWORD dwAlphaDestConstBitDepth;
union
{
        DWORD           dwAlphaDestConst;
        LPDIRECTDRAWSURFACE lpDDSAlphaDest;
};
    DWORD dwAlphaSrcConstBitDepth;
union
{
        DWORD           dwAlphaSrcConst;
        LPDIRECTDRAWSURFACE lpDDSAlphaSrc;
};
    DDCOLORKEY dckDestColorkey;
    DDCOLORKEY dckSrcColorkey;

    DWORD dwDDFX;
    DWORD dwFlags;
} DDOVERLAYFX,FAR *LPDDOVERLAYFX;
```

Members

dwSize
Size of the structure, in bytes. This members must be initialized before the structure is used.

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

dwAlphaEdgeBlend
Constant to use as the alpha for an edge blend.

dwReserved
Reserved for future use.

dwAlphaDestConstBitDepth
Bit depth used to specify the alpha constant for a destination.

dwAlphaDestConst
Constant to use as the alpha channel for a destination.

lpDDSAlphaDest
Address of a surface to use as the alpha channel for a destination.

dwAlphaSrcConstBitDepth
Bit depth used to specify the alpha constant for a source.

dwAlphaSrcConst
Constant to use as the alpha channel for a source.

lpDDSAlphaSrc
Address of a surface to use as the alpha channel for a source.

dckDestColorkey
Destination color key override.

dckSrcColorkey
Source color key override.

dwDDFX
Overlay FX flags.

DDOVERFX_ARITHSTRETCHY
If stretching, use arithmetic stretching along the y-axis for this overlay.

DDOVERFX_MIRRORLEFTRIGHT
Mirror the overlay around the vertical axis.

DDOVERFX_MIRRORUPDOWN
Mirror the overlay around the horizontal axis.

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

DDPIXELFORMAT

The **DDPIXELFORMAT** structure describes the pixel format of a DirectDrawSurface object for the **IDirectDrawSurface3::GetPixelFormat** method.

```
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;
        DWORD dwRGBZBitMask;
        DWORD dwYUVZBitMask;
    };
} DDPIXELFORMAT, FAR* LPDDPIXELFORMAT;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwFlags

Optional control flags.

DDPF_ALPHA

The pixel format describes an alpha-only surface.

DDPF_ALPHAPIXELS

The surface has alpha channel information in the pixel format.

DDPF_COMPRESSED

The surface will accept pixel data in the specified format and compress it

during the write operation.

DDPF_FOURCC

The **dwFourCC** member is valid and contains a FOURCC code describing a non-RGB pixel format..

DDPF_PALETTEINDEXED1

DDPF_PALETTEINDEXED2

DDPF_PALETTEINDEXED4

DDPF_PALETTEINDEXED8

The surface is 1-, 2-, 4-, or 8-bit color indexed.

DDPF_PALETTEINDEXEDTO8

The surface is 1-, 2-, or 4-bit color indexed to an 8-bit palette.

DDPF_RGB

The RGB data in the pixel format structure is valid.

DDPF_RGBTOYUV

The surface will accept RGB data and translate it during the write operation to YUV data. The format of the data to be written will be contained in the pixel format structure. The DDPF_RGB flag will be set.

DDPF_YUV

The YUV data in the pixel format structure is valid.

DDPF_ZBUFFER

The pixel format describes a z-buffer-only surface.

DDPF_ZPIXELS

The surface is in RGBZ format.

dwFourCC

FourCC code. For more information see, Four Character Codes (FOURCC).

dwRGBBitCount

RGB bits per pixel (4, 8, 16, 24, or 32).

dwYUVBitCount

YUV bits per pixel (4, 8, 16, 24, or 32)

dwZBufferBitDepth

Z-buffer bit depth (8, 16, or 24). 32-bit z-buffers are not supported.

dwAlphaBitDepth

Alpha channel bit depth (1, 2, 4, or 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 and **dwYUVAlphaBitMask**
Masks for alpha channel.

dwRGBZBitMask and **dwYUVZBitMask**
Masks for z channel.

DDSCAPS

The **DDSCAPS** structure defines the capabilities of a DirectDrawSurface object. This structure is part of the **DDCAPS** structure that is used to describe the capabilities of the DirectDraw object.

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

Members

dwCaps
Capabilities of the surface. One or more of the following flags:

DDSCAPS_3D
Unsupported. Use the DDSCAPS_3DDEVICE instead.

DDSCAPS_3DDEVICE
Indicates that this surface can be used for 3-D rendering. Applications can use this flag to ensure that a device that can only render to a certain heap has off-screen surfaces allocated from the correct heap. If this flag is set for a heap, the surface is not allocated from that heap.

DDSCAPS_ALLOCONLOAD
Indicates that memory for the surface is not allocated until the surface is loaded by using the **IDirect3DTexture::Load** method.

DDSCAPS_ALPHA
Indicates that this surface contains alpha-only information.

DDSCAPS_BACKBUFFER
Indicates that this surface is the back buffer of a surface flipping structure. Typically, this capability is set by the **IDirectDraw2::CreateSurface** method when the DDSCAPS_FLIP flag is used. Only the surface that immediately precedes the DDSCAPS_FRONTBUFFER surface has this capability set. The other surfaces are identified as back buffers by the presence of the

DDSCAPS_FLIP flag, their attachment order, and the absence of the DDSCAPS_FRONTBUFFER and DDSCAPS_BACKBUFFER capabilities. If this capability is sent to the **IDirectDraw2::CreateSurface** method, a stand-alone back buffer is being created. After this method is called, this surface could be attached to a front buffer, another back buffer, or both to form a flipping surface structure. For more information, see **IDirectDrawSurface3::AddAttachedSurface**. DirectDraw supports an arbitrary number of surfaces in a flipping structure.

DDSCAPS_COMPLEX

Indicates that a complex surface is being described. A complex surface results in the creation of more than one surface. The additional surfaces are attached to the root surface. The complex structure can be destroyed only by destroying the root.

DDSCAPS_FLIP

Indicates that this surface is a part of a surface flipping structure. When this capability is passed to the **IDirectDraw2::CreateSurface** method, a front buffer and one or more back buffers are created. DirectDraw sets the DDSCAPS_FRONTBUFFER bit on the front-buffer surface and the DDSCAPS_BACKBUFFER bit on the surface adjacent to the front-buffer surface. The **dwBackBufferCount** member of the **DDSURFACEDESC** structure must be set to at least 1 in order for the method call to succeed. The DDSCAPS_COMPLEX capability must always be set when creating multiple surfaces by using the **IDirectDraw2::CreateSurface** method.

DDSCAPS_FRONTBUFFER

Indicates that this surface is the front buffer of a surface flipping structure. This flag is typically set by the **IDirectDraw2::CreateSurface** method when the DDSCAPS_FLIP capability is set. If this capability is sent to the **IDirectDraw2::CreateSurface** method, a stand-alone 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 by using **IDirectDrawSurface3::AddAttachedSurface**.

DDSCAPS_HWCODEC

Indicates that this surface should be able to have a stream decompressed to it by the hardware.

DDSCAPS_LIVEVIDEO

Indicates that this surface should be able to receive live video.

DDSCAPS_LOCALVIDMEM

Indicates that this surface exists in true, local video memory rather than non-local video memory. If this flag is specified then DDSCAPS_VIDEOMEMORY must be specified as well. This flag cannot be used with the DDSCAPS_NONLOCALVIDMEM flag.

DDSCAPS_MIPMAP

Indicates that this surface is one level of a mipmap. This surface will be attached to other DDSCAPS_MIPMAP surfaces to form the mipmap. This can be done explicitly by creating a number of surfaces and attaching them by using the **IDirectDrawSurface3::AddAttachedSurface** method, or implicitly by the **IDirectDraw2::CreateSurface** method. If this capability is set, DDSCAPS_TEXTURE must also be set.

DDSCAPS_MODEX

Indicates that this surface is a 320×200 or 320×240 Mode X surface.

DDSCAPS_NONLOCALVIDMEM

Indicates that this surface exists in nonlocal video memory rather than true, local video memory. If this flag is specified, then DDSCAPS_VIDEOMEMORY flag must be specified as well. This cannot be used with the DDSCAPS_LOCALVIDMEM flag.

DDSCAPS_OFFSCREENPLAIN

Indicates that this surface is any off-screen surface that is not an overlay, texture, z-buffer, front-buffer, back-buffer, or alpha surface. It is used to identify plain surfaces.

DDSCAPS_OPTIMIZED

Not currently implemented.

DDSCAPS_OVERLAY

Indicates that this surface is an overlay. It may or may not be directly visible depending on whether it is currently being overlaid onto the primary surface. DDSCAPS_VISIBLE can be used to determine if it is being overlaid at the moment.

DDSCAPS_OWNDC

Indicates that this surface will have a device context (DC) association for a long period.

DDSCAPS_PALETTE

Indicates that this device driver allows unique DirectDrawPalette objects to be created and attached to this surface.

DDSCAPS_PRIMARYSURFACE

Indicates the surface is the primary surface. It represents what is visible to the user at the moment.

DDSCAPS_PRIMARYSURFACELEFT

Indicates that this surface is the primary surface for the left eye. It represents what is visible to the user's left eye at the moment. When this surface is created, the surface with the DDSCAPS_PRIMARYSURFACE capability represents what is seen by the user's right eye.

DDSCAPS_STANDARDVGAMODE

Indicates that this surface is a standard VGA mode surface, and not a

Mode X surface. This flag cannot be used in combination with the DDSCAPS_MODEX flag.

DDSCAPS_SYSTEMMEMORY

Indicates that this surface memory was allocated in system memory.

DDSCAPS_TEXTURE

Indicates that this surface can be used as a 3-D texture. It does not indicate whether the surface is being used for that purpose.

DDSCAPS_VIDEOMEMORY

Indicates that this surface exists in display memory.

DDSCAPS_VIDEOPORT

Indicates that this surface can receive data from a video port.

DDSCAPS_VISIBLE

Indicates that changes made to this surface are immediately visible. It is always set for the primary surface, as well as for overlays while they are being overlaid and texture maps while they are being textured.

DDSCAPS_WRITEONLY

Indicates that only write access is permitted to the surface. Read access from the surface may generate a general protection (GP) fault, but the read results from this surface will not be meaningful.

DDSCAPS_ZBUFFER

Indicates that this surface is the z-buffer. The z-buffer contains information that cannot be displayed. Instead, it contains bit-depth information that is used to determine which pixels are visible and which are obscured.

DDSURFACEDESC

The **DDSURFACEDESC** structure contains a description of the surface to be created. This structure is passed to the **IDirectDraw2::CreateSurface** method. The relevant members differ for each potential type of surface.

```
typedef struct _DDSURFACEDESC {
    DWORD        dwSize;
    DWORD        dwFlags;
    DWORD        dwHeight;
    DWORD        dwWidth;
    union
    {
        LONG     lPitch;
        DWORD    dwLinearSize;
    };
    DWORD        dwBackBufferCount;
    union
```

```
    {
        DWORD       dwMipMapCount;
        DWORD       dwZBufferBitDepth;
        DWORD       dwRefreshRate;
    };
    DWORD           dwAlphaBitDepth;
    DWORD           dwReserved;
    LPVOID          lpSurface;
    DDCOLORKEY      ddckCKDestOverlay;
    DDCOLORKEY      ddckCKDestBlt;
    DDCOLORKEY      ddckCKSrcOverlay;
    DDCOLORKEY      ddckCKSrcBlt;
    DDPIXELFORMAT   ddpfPixelFormat;
    DDSCAPS         ddsCaps;
} DDSURFACEDESC;
```

Members

dwSize

Size of the structure, in bytes. This member must be initialized before the structure is used.

dwFlags

Optional control flags. One or more of the following flags:

DDSD_ALL

Indicates that all input members are valid.

DDSD_ALPHABITDEPTH

Indicates that the **dwAlphaBitDepth** member is valid.

DDSD_BACKBUFFERCOUNT

Indicates that the **dwBackBufferCount** member is valid.

DDSD_CAPS

Indicates that the **ddsCaps** member is valid.

DDSD_CKDESTBLT

Indicates that the **ddckCKDestBlt** member is valid.

DDSD_CKDESTOVERLAY

Indicates that the **ddckCKDestOverlay** member is valid.

DDSD_CKSRCBLT

Indicates that the **ddckCKSrcBlt** member is valid.

DDSD_CKSRCOVERLAY

Indicates that the **ddckCKSrcOverlay** member is valid.

DDSD_HEIGHT

Indicates that the **dwHeight** member is valid.

DDSD_LINEARSIZE
Not used.

DDSD_LPSURFACE
Indicates that the **lpSurface** member is valid.

DDSD_MIPMAPCOUNT
Indicates that the **dwMipMapCount** member is valid.

DDSD_PITCH
Indicates that the **lPitch** member is valid.

DDSD_PIXELFORMAT
Indicates that the **ddpfPixelFormat** member is valid.

DDSD_REFRESHRATE
Indicates that the **dwRefreshRate** member is valid.

DDSD_WIDTH
Indicates that the **dwWidth** member is valid.

DDSD_ZBUFFERBITDEPTH
Indicates that the **dwZBufferBitDepth** member is valid.

dwHeight and **dwWidth**
Dimensions of the surface to be created, in pixels.

lPitch
Distance, in bytes, to the start of next line. When used with the **IDirectDrawSurface3::GetSurfaceDesc** method, this is a return value. When used with the **IDirectDrawSurface3::SetSurfaceDesc** method, this is an input value that must be a DWORD multiple.

dwLinearSize
Not currently used.

dwBackBufferCount
Number of back buffers.

dwMipMapCount
Number of mipmap levels.

dwZBufferBitDepth
Depth of z-buffer. 32-bit z-buffers are not supported.

dwRefreshRate
Refresh rate (used when the display mode is described). The value of 0 indicates an adapter fault.

dwAlphaBitDepth
Depth of alpha buffer.

dwReserved
Reserved.

lpSurface

Address of the associated surface memory. When calling **IDirectDrawSurface3::Lock**, this member is a valid pointer to surface memory. When calling **IDirectDrawSurface3::SetSurfaceDesc**, this member is a pointer to system memory that the caller explicitly allocates for the DirectDrawSurface object.

ddckCKDestOverlay
DDCOLORKEY structure that describes the destination color key to be used for an overlay surface.

ddckCKDestBlt
DDCOLORKEY structure that describes the destination color key for blit operations.

ddckCKSrcOverlay
DDCOLORKEY structure that describes the source color key to be used for an overlay surface.

ddckCKSrcBlt
DDCOLORKEY structure that describes the source color key for blit operations.

ddpfPixelFormat
DDPIXELFORMAT structure that describes the surface's pixel format.

ddsCaps
DDSCAPS structure containing the surface's capabilities.

DDVIDEOPORTBANDWIDTH

The **DDVIDEOPORTBANDWIDTH** structure describes the bandwidth characteristics of an overlay surface when used with a particular video port and pixel format configuration. This structure is used with the **IDirectDrawVideoPort::GetBandwidthInfo** method.

```
typedef struct _DDVIDEOPORTBANDWIDTH {
    DWORD dwSize;          // Size of the DDVIDEOPORTBANDWIDTH structure
    DWORD dwCaps;          // Caps flags
    DWORD dwOverlay;       // Zoom factor at which overlay is supported
    DWORD dwColorkey;      // Zoom factor at which overlay w/ colorkey is supported
    DWORD dwYInterpolate;  // Zoom factor at which overlay w/ Y interpolation is supported
    DWORD dwYInterpAndColorkey;  // Zoom factor at which overlay w/ Y interpolation and
colorkeying is supported
    DWORD dwReserved1;     // Reserved for future use - set to zero
    DWORD dwReserved2;     // Reserved for future use - set to zero
} DDVIDEOPORTBANDWIDTH,*LPDDVIDEOPORTBANDWIDTH;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before use.

dwCaps
Flag values specifying device dependency. This member can be one of the following values.

DDVPBCAPS_DESTINATION	This device's capabilities are described in terms of the overlay's minimum stretch factor. Bandwidth information provided for this device refers to the destination overlay size.
DDVPBCAPS_SOURCE	This device's capabilities are described in terms of the required source overlay size. Bandwidth information provided for this device refers to the source overlay size.

dwOverlay
Stretch factor or overlay source size at which an overlay is supported multiplied by 1000. For example 1.3 = 1300, or .75 = 750.

dwColorkey
Stretch factor or overlay source size at which an overlay with color keying is supported multiplied by 1000. For example 1.3 = 1300, or .75 = 750.

dwYInterpolate
Stretch factor or overlay source size at which an overlay with Y-axis interpolation is supported multiplied by 1000. For example 1.3 = 1300, or .75 = 750.

dwYInterpolateAndColorkey
Stretch factor or overlay source size at which an overlay with Y-axis interpolation and color keying is supported multiplied by 1000. For example 1.3 = 1300, or .75 = 750.

dwReserved1 and **dwReserved2**
Reserved; set to zero.

Remarks

When DDVPBCAPS_DESTINATION is specified, the stretch factors described in the other members describe the minimum stretch factor required to display an overlay with the dimensions given when calling the **GetBandwidthInfo** method. Stretch factor values under 1000 mean that the video port is capable of shrinking an overlay when displayed, and values over 1000 mean that the overlay must be stretched larger than their source to be displayed.

When DDVPBCAPS_SOURCE is specified, the stretch factors described in the other members describe how much you must shrink the overlay source in order for it to be displayed. In this case, the best possible value is 1000, meaning that no shrinking is required. Smaller values tell you that the source rectangle you specified when calling **GetBandwidthInfo** were too large and must be smaller. For example, if the stretch factor is 750 and you specified 320 pixels for the *dwWidth* parameter, then you will not be able to display the overlay at that size. To successfully display the overlay, you must use a source rectangle 240 pixels wide to successfully display the overlay.

DDVIDEOPORTCAPS

The **DDVIDEOPORTCAPS** structure describes the capabilities and alignment restrictions of a video port. This structure is used with the **IDDVideoPortContainer::EnumVideoPorts** method.

```
typedef struct _DDVIDEOPORTCAPS {
    DWORD dwSize;          // Size of the DDVIDEOPORTCAPS structure
    DWORD dwFlags;         // Indicates which fields contain data
    DWORD dwMaxWidth;      // Max width of the video port field
    DWORD dwMaxVBIWidth;   // Max width of the VBI data
    DWORD dwMaxHeight;     // Max height of the video port field
    DWORD dwVideoPortID;   // Video port ID (0 - (dwMaxVideoPorts -1))
    DWORD dwCaps;          // Video port capabilities
    DWORD dwFX;            // More video port capabilities
    DWORD dwNumAutoFlipSurfaces;         // Number of autoflippable surfaces
    DWORD dwAlignVideoPortBoundary;      // Byte restriction of placement within the surface
    DWORD dwAlignVideoPortPrescaleWidth; // Byte restriction of width after prescaling
    DWORD dwAlignVideoPortCropBoundary;  // Byte restriction of left cropping
    DWORD dwAlignVideoPortCropWidth;     // Byte restriction of cropping width
    DWORD dwPreshrinkXStep;          // Width can be shrunk in steps of 1/x
    DWORD dwPreshrinkYStep;          // Height can be shrunk in steps of 1/x
    DWORD dwNumVBIAutoFlipSurfaces;  // Number of VBI autoflippable surfaces
    DWORD dwReserved1;               // Reserved for future use
    DWORD dwReserved2;               // Reserved for future use
} DDVIDEOPORTCAPS, *LPDDVIDEOPORTCAPS;
```

Members

dwSize

Size of the structure, in bytes. This must be initialized before use.

dwFlags

Flag values indicating the fields that contain valid data. The following flags are defined.

DDVD_WIDTH	The **dwMaxWidth** member is valid.
DDVPD_HEIGHT	The **dwMaxHeight** member is valid.
DDVPD_ID	The **dwVideoPortID** member is valid.
DDVPD_CAPS	The **dwCaps** member is valid.
DDVPD_FX	The **dwFX** member is valid.
DDVPD_AUTOFLIP	The **dwNumAutoFlipSurfaces** member is valid.
DDVPD_ALIGN	The **dwAlignVideoPortBoundary**, **dwAlignVideoPortPrescaleWidth**,**dwAlignVideoPortCropBoundary,** and **dwAlignVideoPortCropWidth** are valid.

dwMaxWidth
Maximum width of the video port field.

dwMaxVBIWidth
Maximum width of the VBI data.

dwMaxHeight
Maximum height of the video port field.

dwVideoPortID
Zero based index identifying the video port.

dwCaps
Video port capabilities.

DDVPCAPS_AUTOFLIP	Flip can be performed automatically to avoid tearing when a VREF occurs. If the data is being interleaved in memory, it will flip on every other VREF.
DDVPCAPS_INTERLACED	Supports interlaced video.
DDVPCAPS_NONINTERLACED	Supports non-interlaced video.
DDVPCAPS_READBACKFIELD	Supports the **IDirectDrawVideoPort::GetFieldPolarity** method.
DDVPCAPS_READBACKLINE	Supports the **IDirectDrawVideoPort::GetVideoLine** method.
DDVPCAPS_SHAREABLE	Supports two genlocked video streams that share the video port, where one stream uses the even fields and the other uses the odd fields. Separate parameters (including address, scaling, cropping, etc.) are maintained for both fields.
DDVPCAPS_SKIPEVENFIELDS	Even fields of video can be automatically discarded.
DDVPCAPS_SKIPODDFIELDS	Odd fields of video can be automatically discarded.
DDVPCAPS_SYNCMASTER	Can drive the graphics sync (refresh rate) based on the video port sync.
DDVPCAPS_SYSTEMMEMORY	Capable of writing to surfaces created in system memory.
DDVPCAPS_VBISURFACE	Data within the VBI can be written to a different surface.
DDVPCAPS_COLORCONTROL	Can perform color control operations on incoming data before writing to the frame buffer.
DDVPCAPS_OVERSAMPLEDVBI	Can accept VBI data in a different format or width than the regular video data.

dwFX
Additional video port capabilities.

DDVPFX_CROPTOPDATA	Limited cropping is available to crop VBI data.
DDVPFX_CROPX	Incoming data can be cropped in the x direction before it is written to the surface.
DDVPFX_CROPY	Incoming data can be cropped in the y direction before it is

	written to the surface.
DDVPFX_INTERLEAVE	Supports interleaving interlaced fields in memory.
DDVPFX_MIRRORLEFTRIGHT	Supports mirroring left to right as the video data is written into the frame buffer.
DDVPFX_MIRRORUPDOWN	Supports mirroring top to bottom as the video data is written into the frame buffer.
DDVPFX_PRESHRINKX	Data can be arbitrarily shrunk in the x direction before it is written to the surface.
DDVPFX_PRESHRINKY	Data can be arbitrarily shrunk in the y direction before it is written to the surface.
DDVPFX_PRESHRINKXB	Data can be binary shrunk (1/2, 1/4, 1/8, etc.) in the x direction before it is written to the surface.
DDVPFX_PRESHRINKYB	Data can be binary shrunk (1/2, 1/4, 1/8, etc.) in the y direction before it is written to the surface.
DDVPCAPS_PRESHRINKXS	Data can be shrunk in the x direction by increments of *1/x*, where *x* is specified in the **dwShrinkXStep** member.
DDVPCAPS_PRESHRINKYS	Data can be shrunk in the y direction by increments of *1/y*, where *y* is specified in the **dwShrinkYStep**
DDVPFX_PRESTRETCHX	Data can be arbitrarily stretched in the x direction before it is written to the surface.
DDVPFX_PRESTRETCHY	Data can be arbitrarily stretched in the y direction before it is written to the surface.
DDVPFX_PRESTRETCHXN	Data can be integer stretched in the x direction before it is written to the surface. (1x, 2x, 3x, etc.)
DDVPFX_PRESTRETCHYN	Data can be integer stretched in the y direction before it is written to the surface. (1x, 2x, 3x, etc.)
DDVPFX_VBICONVERT	Data within the VBI can be converted independently of the remaining video data.
DDVPFX_VBINOSCALE	Scaling can be disabled for data within the VBI.
DDVPFX_IGNOREVBIXCROP	The video port can ignore the left and right cropping coordinates when cropping oversampled VBI data.

dwNumAutoFlipSurfaces
Number of autoflippable surfaces supported by the video port.

dwAlignVideoPortBoundary
Byte restriction of placement within the surface.

dwAlignVideoPortPrescaleWidth
Byte restriction of width after prescaling.

dwAlignVideoPortCropBoundary
Byte restriction of left cropping.

dwAlignVideoPortCropWidth
Byte restriction of cropping width.

dwPreshrinkXStep
Width can be shrunk in the x direction in steps of 1/**dwPreshrinkXStep**.

dwPreshrinkYStep
Height can be shrunk in the y direction in steps of *1/***dwPreshrinkYStep**.

dwNumVBIAutoFlipSurfaces
Number of autoflipping surfaces capable of receiving data transmitted during the vertical blanking interval (VBI) independent from the remainder of the video stream. When constructing the autoflip chain, the number of VBI surfaces must equal the number of surfaces receiving the remainder of the video data.

dwReserved1 and **dwReserved2**
Reserved; set to zero.

DDVIDEOPORTCONNECT

The DDVIDEOPORTCONNECT structure describes a video port connection. This structure is used with the **IDDVideoPortContainer::GetVideoPortConnectInfo** method.

```
typedef struct _DDVIDEOPORTCONNECT{
    DWORD dwSize;          // Size of the DDVIDEOPORTCONNECT structure
    DWORD dwPortWidth;     // Width of the video port
    GUID  guidTypeID;      // Description of video port connection
    DWORD dwFlags;         // Connection flags
    DWORD dwReserved1;     // Reserved, set to zero.
} DDVIDEOPORTCONNECT,*LPDDVIDEOPORTCONNECT;
```

Members

dwSize
Size of the structure, in bytes. This member must be initialized before use.

dwPortWidth
Width of the video port. This value represents the number of physical pins on the video port, not the width of a surface in memory. This member must always be set, even when the value in the **guidTypeID** member assumes a certain size.

guidTypeID
A GUID that describes the sync characteristics of the video port. The following port types are predefined:

DDVPTYPE_E_HREFH_VREFH	External syncs where HREF is active high and VREF is active high.
DDVPTYPE_E_HREFH_VREFL	External syncs where HREF is active high and VREF is active low.
DDVPTYPE_E_HREFL_VREFH	External syncs where HREF is active low and VREF is active high.
DDVPTYPE_E_HREFL_VREFL	External syncs where HREF is active low and VREF is active

	low.
DDVPTYPE_CCIR656	Sync information is embedded in the data stream according to th CCIR656 spec.
DDVPTYPE_BROOKTREE	Sync information is embedded in the data stream using the Brooktree definition.
DDVPTYPE_PHILIPS	Sync information is embedded in the data stream using the Philips definition.

dwFlags

Flags describing the capabilities of the video-port connection. This member can be set by DirectDraw when connection information is being retrieved or by the client when connection information is being set. This member can be a combination of the following flags.

DDVPCONNECT_DOUBLECLOCK	Indicates that the video port either supports double-clocking data or should double-clock data. This flag is only valid with an external sync.
DDVPCONNECT_VACT	Indicates that the video port either supports using an external VACT signal or should use the external VACT signal. This flag is only valid with an external sync.
DDVPCONNECT_INVERTPOLARITY	Indicates that the video port is capable of inverting the field polarities or is to invert field polarities. When a video port inverts field polarities, it treats even fields as odd fields and vice versa.
DDVPCONNECT_DISCARDSVREFDATA	The video port discards any data written during the VREF period, causing it to not be written to the frame buffer. This flag is read-only.
DDVPCONNECT_HALFLINE	The video port will write half lines into the frame buffe sometimes causing the data to be displayed incorrectly. This flag is read-only.
DDVPCONNECT_INTERLACED	Indicates that the signal is interlaced. This flag is only used by the client when creating a video port object.
DDVPCONNECT_SHAREEVEN	The physical video port is shareable, and that this video port object will use the even fields. This flag is only used by the client when creating the video port object.
DDVPCONNECT_SHAREODD	The physical video port is shareable, and that this video port object will use the odd fields. This flag is only used by the client when creating the video port object.

dwReserved1

Reserved; set to zero.

Remarks

This structure is used independently and as a member of the **DDVIDEOPORTDESC** structure.

DDVIDEOPORTDESC

The **DDVIDEOPORTDESC** structure describes a video-port object to be created. This structure is used with the **IDDVideoPortContainer::CreateVideoPort** method.

```
typedef struct _DDVIDEOPORTDESC {
    DWORD dwSize;               // Size of the DDVIDEOPORTDESC structure.
    DWORD dwFieldWidth;         // Width of the video port field.
    DWORD dwVBIWidth;           // Width of the VBI data.
    DWORD dwFieldHeight;        // Height of the video port field.
    DWORD dwMicrosecondsPerField; // Microseconds per video field.
    DWORD dwMaxPixelsPerSecond;   // Maximum pixel rate per second.
    DWORD dwVideoPortID;        // Video port ID (0 - (dwMaxVideoPorts -1)).
    DWORD dwReserved1;          // Reserved for future use - set to zero.
    DDVIDEOPORTCONNECT VideoPortType;  // Description of video port connection.
    DWORD dwReserved2;          // Reserved for future use - set to zero.
    DWORD dwReserved3;          // Reserved for future use - set to zero.
} DDVIDEOPORTDESC, *LPDDVIDEOPORTDESC;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before use.

dwFieldWidth
Width of incoming video stream, in pixels.

dwVBIWidth
Width of the VBI data in the incoming video stream, in pixels.

dwFieldHeight
Field height for fields in the incoming video stream, in scan lines.

dwMicrosecondsPerField
Time interval, in microseconds, between live video VREF periods. This number should be rounded up to the nearest microsecond.

dwVideoPortID
The zero-based ID of the physical video port to be used.

dwReserved1
Reserved; set to zero.

VideoPortType
A **DDVIDEOPORTCONNECT** structure describing the connection characteristics of the video port.

dwReserved2 and **dwReserved3**
Reserved; set to zero.

DDVIDEOPORTINFO

The DDVIDEOPORTINFO structure describes the transfer of video data to a surface. This structure is used with the **IDirectDrawVideoPort::StartVideo** method.

```
typedef struct _DDVIDEOPORTINFO{
    DWORD dwSize;            // Size of the structure.
    DWORD dwOriginX;         // Placement of the video data within the surface.
    DWORD dwOriginY;         // Placement of the video data within the surface.
    DWORD dwVPFlags;         // Video port options.
    RECT rCrop;              // Cropping rectangle (optional).
    DWORD dwPrescaleWidth;   // Pre-scaling/zooming in the X direction (optional).
    DWORD dwPrescaleHeight;  // Pre-scaling/zooming in the Y direction (optional).
    LPDDPIXELFORMAT lpddpfInputFormat;     // Video format written to the video port.
    LPDDPIXELFORMAT lpddpfVBIInputFormat;  // Input format of the VBI data.
    LPDDPIXELFORMAT lpddpfVBIOutputFormat; // Output format of the data.
    DWORD dwVBIHeight;       // Lines of data within the vertical blanking interval.
    DWORD dwReserved1;       // Reserved for future use - set to zero.
    DWORD dwReserved2;       // Reserved for future use - set to zero.
} DDVIDEOPORTINFO,*LPDDVIDEOPORTINFO;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before use.

dwOriginX and **dwOriginY**
X and y coordinates for the origin of the video data in the surface.

dwVPFlags
Video port options.

DDVP_AUTOFLIP	Perform automatic flipping. For more information, se Auto-flipping.
DDVP_CONVERT	Perform conversion using the information in the **ddpfOutputFormat** member.
DDVP_CROP	Perform cropping using the rectangle specified by the **rCrop** member.
DDVP_INTERLEAVE	Interlaced fields should be interleaved in memory.
DDVP_MIRRORLEFTRIGHT	Mirror image data from left to right as it is written int frame buffer.
DDVP_MIRRORUPDOWN	Mirror image data from top to bottom as it is written the frame buffer.
DDVP_PRESCALE	Perform pre-scaling or pre-zooming based on the valu the **dwPrescaleHeight** and **dwPrescaleWidth** memb
DDVP_SKIPEVENFIELDS	Ignore input of even fields.
DDVP_SKIPODDFIELDS	Ignore input of odd fields.

DDVP_SYNCMASTER	Indicates that the video port VREF should drive the graphics VREF, locking the refresh rate to the video p
DDVP_VBICONVERT	The **ddpfVBIOutputFormat** member contains data t should be used to convert VBI data.
DDVP_VBINOSCALE	VBI data should not be scaled.
DDVPCONNECT_OVERRIDEBOBWEAVE	Override automatic display method chosen by the dri using only the display method set by the caller when creating the overlay surface.
DDVPFX_IGNOREVBIXCROP	Indicates that the video port should ignore left and rig cropping coordinates when cropping oversampled VB data.

rCrop
Cropping rectangle. This member is optional.

dwPrescaleWidth
Pre-scaling or zooming in the x direction. This member is optional.

dwPrescaleHeight
Pre-scaling or zooming in the y direction. This member is optional.

ddpfInputFormat
A **DDPIXELFORMAT** structure describing the pixel format to be written to the video port. This will often be identical to the surface's pixel format, but can differ if the video port is to perform conversion.

ddpfVBIInputFormat and **ddpfVBIOutputFormat**
DDPIXELFORMAT structures describing the input and output pixel formats of the data within the vertical blanking interval.

dwVBIHeight
The amount of data within the vertical blanking interval, in scan lines.

dwReserved1 and **dwReserved2**
Reserved; set to zero.

DDVIDEOPORTSTATUS

The **DDVIDEOPORTSTATUS** structure describes the status of a video-port object. This structure is used with the **IDDVideoPortContainer::QueryVideoPortStatus** method.

```
typedef struct _DDVIDEOPORTSTATUS {
    DWORD dwSize;        // size of the structure
    BOOL  bInUse;        // TRUE if video port is currently being used
    DWORD dwFlags;       // not used
    DWORD dwReserved1;   // reserved for future use
    DDVIDEOPORTCONNECT VideoPortType;    // information about the connection
    DWORD dwReserved2;   // reserved for future use
    DWORD dwReserved3;   // reserved for future use
```

```
} DDVIDEOPORTSTATUS, *LPDDVIDEOPORTSTATUS;
```

Members

dwSize
Size of this structure, in bytes. This member must be initialized before use.

bInUse
Value indicating the current status of the video port. This member is TRUE if the video port is currently being used, and FALSE otherwise.

dwFlags
Not currently used.

dwReserved1
Reserved; set to zero.

VideoPortType
A **DDVIDEOPORTCONNECT** structure that receives information about the video-port connection.

dwReserved2 and **dwReserved3**
Reserved; set to zero.

Return Values

Errors are represented by negative values and cannot be combined. This table lists the values that can be returned by all methods of the **IDirectDraw2**, **IDirectDrawSurface3**, **IDirectDrawPalette**, **IDirectDrawClipper** and **IDirectDrawVideoPort** interfaces. For a list of the error codes that each method can return, see the method description.

DD_OK
The request completed successfully.

DDERR_ALREADYINITIALIZED
The object has already been initialized.

DDERR_BLTFASTCANTCLIP
A DirectDrawClipper object is attached to a source surface that has passed into a call to the **IDirectDrawSurface3::BltFast** method.

DDERR_CANNOTATTACHSURFACE
A surface cannot be attached to another requested surface.

DDERR_CANNOTDETACHSURFACE
A surface cannot be detached from another requested surface.

DDERR_CANTCREATEDC
Windows cannot create any more device contexts (DCs).

DDERR_CANTDUPLICATE
Primary and 3-D surfaces, or surfaces that are implicitly created, cannot be

duplicated.

DDERR_CANTLOCKSURFACE

Access to this surface is refused because an attempt was made to lock the primary surface without DCI support.

DDERR_CANTPAGELOCK

An attempt to page lock a surface failed. Page lock will not work on a display-memory surface or an emulated primary surface.

DDERR_CANTPAGEUNLOCK

An attempt to page unlock a surface failed. Page unlock will not work on a display-memory surface or an emulated primary surface.

DDERR_CLIPPERISUSINGHWND

An attempt was made to set a clip list for a DirectDrawClipper object that is already monitoring a window handle.

DDERR_COLORKEYNOTSET

No source color key is specified for this operation.

DDERR_CURRENTLYNOTAVAIL

No support is currently available.

DDERR_DCALREADYCREATED

A device context (DC) has already been returned for this surface. Only one DC can be retrieved for each surface.

DDERR_DEVICEDOESNTOWNSURFACE

Surfaces created by one direct draw device cannot be used directly by another direct draw device.

DDERR_DIRECTDRAWALREADYCREATED

A DirectDraw object representing this driver has already been created for this process.

DDERR_EXCEPTION

An exception was encountered while performing the requested operation.

DDERR_EXCLUSIVEMODEALREADYSET

An attempt was made to set the cooperative level when it was already set to exclusive.

DDERR_GENERIC

There is an undefined error condition.

DDERR_HEIGHTALIGN

The height of the provided rectangle is not a multiple of the required alignment.

DDERR_HWNDALREADYSET

The DirectDraw cooperative level window handle has already been set. It cannot be reset while the process has surfaces or palettes created.

DDERR_HWNDSUBCLASSED

DirectDraw is prevented from restoring state because the DirectDraw cooperative level window handle has been subclassed.

DDERR_IMPLICITLYCREATED

The surface cannot be restored because it is an implicitly created surface.

DDERR_INCOMPATIBLEPRIMARY

The primary surface creation request does not match with the existing primary surface.

DDERR_INVALIDCAPS

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

DDERR_INVALIDCLIPLIST

DirectDraw does not support the provided clip list.

DDERR_INVALIDDIRECTDRAWGUID

The globally unique identifier (GUID) passed to the **DirectDrawCreate** function is not a valid DirectDraw driver identifier.

DDERR_INVALIDMODE

DirectDraw does not support the requested mode.

DDERR_INVALIDOBJECT

DirectDraw received a pointer that was an invalid DirectDraw object.

DDERR_INVALIDPARAMS

One or more of the parameters passed to the method are incorrect.

DDERR_INVALIDPIXELFORMAT

The pixel format was invalid as specified.

DDERR_INVALIDPOSITION

The position of the overlay on the destination is no longer legal.

DDERR_INVALIDRECT

The provided rectangle was invalid.

DDERR_INVALIDSURFACETYPE

The requested operation could not be performed because the surface was of the wrong type.

DDERR_LOCKEDSURFACES

One or more surfaces are locked, causing the failure of the requested operation.

DDERR_MOREDATA

There is more data available than the specified buffer size can hold.

DDERR_NO3D

No 3-D hardware or emulation is present.

DDERR_NOALPHAHW

No alpha acceleration hardware is present or available, causing the failure of

the requested operation.

DDERR_NOBLTHW

No blitter hardware is present.

DDERR_NOCLIPLIST

No clip list is available.

DDERR_NOCLIPPERATTACHED

No DirectDrawClipper object is attached to the surface object.

DDERR_NOCOLORCONVHW

The operation cannot be carried out because no color-conversion hardware is present or available.

DDERR_NOCOLORKEY

The surface does not currently have a color key.

DDERR_NOCOLORKEYHW

The operation cannot be carried out because there is no hardware support for the destination color key.

DDERR_NOCOOPERATIVELEVELSET

A create function is called without the **IDirectDraw2::SetCooperativeLevel** method being called.

DDERR_NODC

No DC has ever been created for this surface.

DDERR_NODDROPSHW

No DirectDraw raster operation (ROP) hardware is available.

DDERR_NODIRECTDRAWHW

Hardware-only DirectDraw object creation is not possible; the driver does not support any hardware.

DDERR_NODIRECTDRAWSUPPORT

DirectDraw support is not possible with the current display driver.

DDERR_NOEMULATION

Software emulation is not available.

DDERR_NOEXCLUSIVEMODE

The operation requires the application to have exclusive mode, but the application does not have exclusive mode.

DDERR_NOFLIPHW

Flipping visible surfaces is not supported.

DDERR_NOGDI

No GDI is present.

DDERR_NOHWND

Clipper notification requires a window handle, or no window handle has been previously set as the cooperative level window handle.

DDERR_NOMIPMAPHW

The operation cannot be carried out because no mipmap texture mapping hardware is present or available.

DDERR_NOMIRRORHW

The operation cannot be carried out because no mirroring hardware is present or available.

DDERR_NONONLOCALVIDMEM

An attempt was made to allocate non-local video memory from a device that does not support non-local video memory.

DDERR_NOOPTIMIZEHW

The device does not support optimized surfaces.

DDERR_NOOVERLAYDEST

The **IDirectDrawSurface3::GetOverlayPosition** method is called on an overlay that the **IDirectDrawSurface3::UpdateOverlay** method has not been called on to establish a destination.

DDERR_NOOVERLAYHW

The operation cannot be carried out because no overlay hardware is present or available.

DDERR_NOPALETTEATTACHED

No palette object is attached to this surface.

DDERR_NOPALETTEHW

There is no hardware support for 16- or 256-color palettes.

DDERR_NORASTEROPHW

The operation cannot be carried out because no appropriate raster operation hardware is present or available.

DDERR_NOROTATIONHW

The operation cannot be carried out because no rotation hardware is present or available.

DDERR_NOSTRETCHHW

The operation cannot be carried out because there is no hardware support for stretching.

DDERR_NOT4BITCOLOR

The DirectDrawSurface object is not using a 4-bit color palette and the requested operation requires a 4-bit color palette.

DDERR_NOT4BITCOLORINDEX

The DirectDrawSurface object is not using a 4-bit color index palette and the requested operation requires a 4-bit color index palette.

DDERR_NOT8BITCOLOR

The DirectDrawSurface object is not using an 8-bit color palette and the requested operation requires an 8-bit color palette.

DDERR_NOTAOVERLAYSURFACE

An overlay component is called for a non-overlay surface.

DDERR_NOTEXTUREHW

The operation cannot be carried out because no texture-mapping hardware is present or available.

DDERR_NOTFLIPPABLE

An attempt has been made to flip a surface that cannot be flipped.

DDERR_NOTFOUND

The requested item was not found.

DDERR_NOTINITIALIZED

An attempt was made to call an interface method of a DirectDraw object created by **CoCreateInstance** before the object was initialized.

DDERR_NOTLOADED

The surface is an optimized surface, but it has not yet been allocated any memory.

DDERR_NOTLOCKED

An attempt is made to unlock a surface that was not locked.

DDERR_NOTPAGELOCKED

An attempt is made to page unlock a surface with no outstanding page locks.

DDERR_NOTPALETTIZED

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

DDERR_NOVSYNCHW

The operation cannot be carried out because there is no hardware support for vertical blank synchronized operations.

DDERR_NOZBUFFERHW

The operation to create a z-buffer in display memory or to perform a blit using a z-buffer cannot be carried out because there is no hardware support for z-buffers.

DDERR_NOZOVERLAYHW

The overlay surfaces cannot be z-layered based on the z-order because the hardware does not support z-ordering of overlays.

DDERR_OUTOFCAPS

The hardware needed for the requested operation has already been allocated.

DDERR_OUTOFMEMORY

DirectDraw does not have enough memory to perform the operation.

DDERR_OUTOFVIDEOMEMORY

DirectDraw does not have enough display memory to perform the operation.

DDERR_OVERLAYCANTCLIP

The hardware does not support clipped overlays.

DDERR_OVERLAYCOLORKEYONLYONEACTIVE

An attempt was made to have more than one color key active on an overlay.

DDERR_OVERLAYNOTVISIBLE

The **IDirectDrawSurface3::GetOverlayPosition** method is called on a hidden overlay.

DDERR_PALETTEBUSY

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

DDERR_PRIMARYSURFACEALREADYEXISTS

This process has already created a primary surface.

DDERR_REGIONTOOSMALL

The region passed to the **IDirectDrawClipper::GetClipList** method is too small.

DDERR_SURFACEALREADYATTACHED

An attempt was made to attach a surface to another surface to which it is already attached.

DDERR_SURFACEALREADYDEPENDENT

An attempt was made to make a surface a dependency of another surface to which it is already dependent.

DDERR_SURFACEBUSY

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

DDERR_SURFACEISOBSCURED

Access to the surface is refused because the surface is obscured.

DDERR_SURFACELOST

Access to the surface is refused because the surface memory is gone. The DirectDrawSurface object representing this surface should have the **IDirectDrawSurface3::Restore** method called on it.

DDERR_SURFACENOTATTACHED

The requested surface is not attached.

DDERR_TOOBIGHEIGHT

The height requested by DirectDraw is too large.

DDERR_TOOBIGSIZE

The size requested by DirectDraw is too large. However, the individual height and width are OK.

DDERR_TOOBIGWIDTH

The width requested by DirectDraw is too large.

DDERR_UNSUPPORTED

The operation is not supported.

DDERR_UNSUPPORTEDFORMAT

The FourCC format requested is not supported by DirectDraw.

DDERR_UNSUPPORTEDMASK

The bitmask in the pixel format requested is not supported by DirectDraw.

DDERR_UNSUPPORTEDMODE

The display is currently in an unsupported mode.

DDERR_VERTICALBLANKINPROGRESS

A vertical blank is in progress.

DDERR_VIDEONOTACTIVE

The video port is not active.

DDERR_WASSTILLDRAWING

The previous blit operation that is transferring information to or from this surface is incomplete.

DDERR_WRONGMODE

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

DDERR_XALIGN

The provided rectangle was not horizontally aligned on a required boundary.

Pixel Format Masks

This section contains information about the pixel formats supported by the hardware-emulation layer (HEL). The following topics are discussed:

- Texture Map Formats
- Off-Screen Surface Formats

Texture Map Formats

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

Pixel format flags	Bit depth	Masks
DDPF_RGB |	1	R: 0x00000000
DDPF_PALETTEINDEXED1		G: 0x00000000
		B: 0x00000000
		A: 0x00000000
DDPF_RGB |	1	R: 0x00000000

DDPF_PALETTEINDEXED1 | DDPF_PALETTEINDEXEDTO8		G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2 | DDPF_PALETTEINDEXEDTO8	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED4	4	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED4 | DDPF_PALETTEINDEXEDTO8	4	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED8	8	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB	8	R: 0x000000E0 G: 0x0000001C

		B: 0x00000003 A: 0x00000000
DDPF_RGB | DDPF_ALPHAPIXELS	16	R: 0x00000F00 G: 0x000000F0 B: 0x0000000F A: 0x0000F000
DDPF_RGB	16	R: 0x0000F800 G: 0x000007E0 B: 0x0000001F A: 0x00000000
DDPF_RGB	16	R: 0x0000001F G: 0x000007E0 B: 0x0000F800 A: 0x00000000
DDPF_RGB	16	R: 0x00007C00 G: 0x000003E0 B: 0x0000001F A: 0x00000000
DDPF_RGB | DDPF_ALPHAPIXELS	16	R: 0x00007C00 G: 0x000003E0 B: 0x0000001F A: 0x00008000
DDPF_RGB	24	R: 0x00FF0000 G: 0x0000FF00 B: 0x000000FF

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

The HEL can create these formats in system memory. The DirectDraw device driver for a 3-D–accelerated display card may create textures of other formats in display memory. Such a driver exports the DDSCAPS_TEXTURE flag to indicate that it can create textures.

Off-Screen Surface Formats

The following table shows the pixel formats for off-screen plain surfaces supported by the DirectX® 5 HEL. The Masks column contains the red, green, blue, and alpha masks for each set of pixel format flags and bit depths.

Pixel format flags	Bit depth	Masks
DDPF_RGB | DDPF_PALETTEINDEXED1	1	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED2	2	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED4	4	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB | DDPF_PALETTEINDEXED8	8	R: 0x00000000 G: 0x00000000 B: 0x00000000 A: 0x00000000
DDPF_RGB	16	R: 0x0000F800 G: 0x000007E0 B: 0x0000001F A: 0x00000000
DDPF_RGB	16	R: 0x00007C00

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

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

Four Character Codes (FOURCC)

DirectDraw utilizes a special set of codes that are four characters in length. These codes, called four character codes or FOURCCs, are stored in file headers of files containing multimedia data such as bitmap images, sound, or video. FOURCCs describe the software technology that was used to produce multimedia data. By implication, they also describe the format of the data itself.

DirectDraw applications use FOURCCs for image color and format conversion. If an application calls the **IDirectDrawSurface3::GetPixelFormat** method to request the pixel format of a surface whose format is not RGB, the **dwFourCC** member of the **DDPIXELFORMAT** structure identifies the FOURCC when the method returns. For more information, see Converting Color and Format.

In addition, the **biCompression** member of the **BITMAPINFOHEADER** structure can be set to a FOURCC to indicate the codec used to compress or decompress an image.

FOURCCs are registered with Microsoft by the vendors of the respective multimedia software technologies. Some common FOURCCs appear in the list below.

FOURCC	Company	Technology Name
AUR2	AuraVision Corporation	AuraVision Aura 2: YUV 422
AURA	AuraVision Corporation	AuraVision Aura 1: YUV 411
CHAM	Winnov, Inc.	MM_WINNOV_CAVIARA_CHAMPAGNE
CVID	Supermac	Cinepak by Supermac
CYUV	Creative Labs, Inc	Creative Labs YUV
FVF1	Iterated Systems, Inc.	Fractal Video Frame
IF09	Intel Corporation	Intel Intermediate YUV9
IV31	Intel Corporation	Indeo 3.1
JPEG	Microsoft Corporation	Still Image JPEG DIB
MJPG	Microsoft Corporation	Motion JPEG Dib Format
MRLE	Microsoft Corporation	Run Length Encoding
MSVC	Microsoft Corporation	Video 1
PHMO	IBM Corporation	Photomotion
RT21	Intel Corporation	Indeo 2.1
ULTI	IBM Corporation	Ultimotion
V422	Vitec Multimedia	24 bit YUV 4:2:2
V655	Vitec Multimedia	16 bit YUV 4:2:2
VDCT	Vitec Multimedia	Video Maker Pro DIB
VIDS	Vitec Multimedia	YUV 4:2:2 CCIR 601 for V422
YU92	Intel Corporation	YUV
YUV8	Winnov, Inc.	MM_WINNOV_CAVIAR_YUV8
YUV9	Intel Corporation	YUV9
YUYV	Canopus, Co., Ltd.	BI_YUYV, Canopus
ZPEG	Metheus	Video Zipper