

DXGI

Article 01/24/2023

Overview of the DXGI technology.

To develop DXGI, you need these headers:

- [dxgi.h](#)
- [dxgi1_2.h](#)
- [dxgi1_3.h](#)
- [dxgi1_4.h](#)
- [dxgi1_5.h](#)
- [dxgi1_6.h](#)
- [dxgicommon.h](#)
- [dxgidebug.h](#)
- [dxgiformat.h](#)

For programming guidance for this technology, see:

- [DXGI](#)

Enumerations

DXGI_ADAPTER_FLAG
Identifies the type of DXGI adapter. (DXGI_ADAPTER_FLAG)
DXGI_ADAPTER_FLAG3
Identifies the type of DXGI adapter. (DXGI_ADAPTER_FLAG3)
DXGI_ALPHA_MODE
Identifies the alpha value, transparency behavior, of a surface.
DXGI_COLOR_SPACE_TYPE
Specifies color space types.
DXGI_COMPUTE_PREEMPTION_GRANULARITY
Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current compute task.

[DXGI_DEBUG_RLO_FLAGS](#)

Flags used with ReportLiveObjects to specify the amount of info to report about an object's lifetime.

[DXGI_FEATURE](#)

Specifies a range of hardware features, to be used when checking for feature support.

[DXGI_FORMAT](#)

Resource data formats, including fully-typed and typeless formats. A list of modifiers at the bottom of the page more fully describes each format type.

[DXGI_FRAME_PRESENTATION_MODE](#)

Indicates options for presenting frames to the swap chain.

[DXGI_GPU_PREFERENCE](#)

The preference of GPU for the app to run on.

[DXGI_GRAPHICS_PREEMPTION_GRANULARITY](#)

Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current graphics rendering task.

[DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAGS](#)

Describes which levels of hardware composition are supported.

[DXGI_HDR_METADATA_TYPE](#)

Specifies the header metadata type.

[DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)

Values that specify categories of debug messages.

[DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)

Values that specify debug message severity levels for an information queue.

[DXGI_MEMORY_SEGMENT_GROUP](#)

Specifies the memory segment group to use.

DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS	Options for swap-chain color space.
DXGI_OFFER_RESOURCE_FLAGS	Specifies flags for the OfferResources1 method.
DXGI_OFFER_RESOURCE_PRIORITY	Identifies the importance of a resource's content when you call the IDXGIDevice2::OfferResources method to offer the resource.
DXGI_OUTDUPL_POINTER_SHAPE_TYPE	Identifies the type of pointer shape.
DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG	Specifies support for overlay color space.
DXGI_OVERLAY_SUPPORT_FLAG	Specifies overlay support to check for in a call to IDXGIOutput3::CheckOverlaySupport.
DXGI_RECLAIM_RESOURCE_RESULTS	Specifies result flags for the ReclaimResources1 method.
DXGI_RESIDENCY	Flags indicating the memory location of a resource.
DXGI_SCALING	Identifies resize behavior when the back-buffer size does not match the size of the target output.
DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG	Specifies color space support for the swap chain.
DXGI_SWAP_CHAIN_FLAG	Options for swap-chain behavior.
DXGI_SWAP_EFFECT	Options for handling pixels in a display surface after calling IDXGISwapChain1::Present1.

Functions

AcquireNextFrame	Indicates that the application is ready to process the next desktop image.
AcquireSync	Using a key, acquires exclusive rendering access to a shared resource.
AddApplicationMessage	Adds a user-defined message to the message queue and sends that message to the debug output.
AddMessage	Adds a debug message to the message queue and sends that message to the debug output.
AddRetrievalFilterEntries	Adds retrieval filters to the top of the retrieval-filter stack.
AddStorageFilterEntries	Adds storage filters to the top of the storage-filter stack.
CheckColorSpaceSupport	Checks the swap chain's support for color space.
CheckFeatureSupport	Used to check for hardware feature support.
CheckHardwareCompositionSupport	Notifies applications that hardware stretching is supported.
CheckInterfaceSupport	Checks whether the system supports a device interface for a graphics component.
CheckOverlayColorSpaceSupport	Checks for overlay color space support.

CheckOverlaySupport	Checks for overlay support.
CheckPresentDurationSupport	Queries the graphics driver for a supported frame present duration corresponding to a custom refresh rate.
ClearRetrievalFilter	Removes a retrieval filter from the top of the retrieval-filter stack.
ClearStorageFilter	Removes a storage filter from the top of the storage-filter stack.
ClearStoredMessages	Clears all messages from the message queue.
CreateDecodeSwapChainForCompositionSurfaceHandle	Creates a YUV swap chain for an existing DirectComposition surface handle. (IDXGIFactoryMedia.CreateDecodeSwapChainForCompositionSurfaceHandle)
CreateDXGIFactory	Creates a DXGI 1.0 factory that you can use to generate other DXGI objects.
CreateDXGIFactory1	Creates a DXGI 1.1 factory that you can use to generate other DXGI objects.
CreateDXGIFactory2	Creates a DXGI 1.3 factory that you can use to generate other DXGI objects.
CreateSharedHandle	Creates a handle to a shared resource. You can then use the returned handle with multiple Direct3D devices.
CreateSoftwareAdapter	Create an adapter interface that represents a software adapter.

CreateSubresourceSurface	Creates a subresource surface object.
CreateSurface	Returns a surface. This method is used internally and you should not call it directly in your application.
CreateSwapChain	Creates a swap chain.
CreateSwapChainForComposition	Creates a swap chain that you can use to send Direct3D content into the DirectComposition API or a Xaml framework to compose in a window.
CreateSwapChainForCompositionSurfaceHandle	Creates a YUV swap chain for an existing DirectComposition surface handle. (IDXGIFactoryMedia.CreateSwapChainForCompositionSurfaceHandle)
CreateSwapChainForCoreWindow	Creates a swap chain that is associated with the CoreWindow object for the output window for the swap chain.
CreateSwapChainForHwnd	Creates a swap chain that is associated with an HWND handle to the output window for the swap chain.
DisableLeakTrackingForThread	Stops tracking leaks for the current thread.
DuplicateOutput	Creates a desktop duplication interface from the IDXGIOutput1 interface that represents an adapter output.
DuplicateOutput1	Allows specifying a list of supported formats for fullscreen surfaces that can be returned by the IDXGIDuplicateOutput object.

[DXGIDeclareAdapterRemovalSupport](#)

Allows a process to indicate that it's resilient to any of its graphics devices being removed.

[DXGIDisableVBlankVirtualization](#)

Disables v-blank virtualization for the process. This virtualization is used by the dynamic refresh rate (DRR) feature by default for all swap chains to maintain a steady virtualized present rate and v-blank cadence from [IDXGIOOutput::WaitForVBlank](#). By disabling virtualization, these APIs will see the changing refresh rate.

[DXGIGetDebugInterface](#)

Retrieves a debugging interface.

[DXGIGetDebugInterface1](#)

Retrieves an interface that Windows Store apps use for debugging the Microsoft DirectX Graphics Infrastructure (DXGI).

[EnableLeakTrackingForThread](#)

Starts tracking leaks for the current thread.

[EnqueueSetEvent](#)

Flushes any outstanding rendering commands and sets the specified event object to the signaled state after all previously submitted rendering commands complete.

[EnumAdapterByGpuPreference](#)

Enumerates graphics adapters based on a given GPU preference.

[EnumAdapterByLuid](#)

Outputs the IDXGIAdapter for the specified LUID.

[EnumAdapters](#)

Enumerates the adapters (video cards).

[EnumAdapters1](#)

Enumerates both adapters (video cards) with or without outputs.

[EnumOutputs](#)

Enumerate adapter (video card) outputs.

EnumWarpAdapter
Provides an adapter which can be provided to D3D12CreateDevice to use the WARP renderer.
FindClosestMatchingMode
Finds the display mode that most closely matches the requested display mode. (IDXGIOutput.FindClosestMatchingMode)
FindClosestMatchingMode1
Finds the display mode that most closely matches the requested display mode. (IDXGIOutput1.FindClosestMatchingMode1)
GetAdapter
Returns the adapter for the specified device.
GetBackgroundColor
Retrieves the background color of the swap chain.
GetBreakOnCategory
Determines whether the break on a message category is turned on or off.
GetBreakOnID
Determines whether the break on a message identifier is turned on or off.
GetBreakOnSeverity
Determines whether the break on a message severity level is turned on or off.
GetBuffer
Accesses one of the swap-chain's back buffers.
GetColorSpace
Gets the color space used by the swap chain.
GetContainingOutput
Get the output (the display monitor) that contains the majority of the client area of the target window.

[GetCoreWindow](#)

Retrieves the underlying CoreWindow object for this swap-chain object.

[GetCreationFlags](#)

Gets the flags that were used when a Microsoft DirectX Graphics Infrastructure (DXGI) object was created.

[GetCurrentBackBufferIndex](#)

Gets the index of the swap chain's current back buffer.

[GetDC](#)

Returns a device context (DC) that allows you to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface using Windows Graphics Device Interface (GDI).

[GetDesc](#)

Gets a DXGI 1.0 description of an adapter (or video card).

[GetDesc](#)

Get a description of the output.

[GetDesc](#)

Get a description of the surface.

[GetDesc](#)

Get a description of the swap chain.

[GetDesc](#)

Retrieves a description of a duplicated output. This description specifies the dimensions of the surface that contains the desktop image.

[GetDesc1](#)

Gets a DXGI 1.1 description of an adapter (or video card).

[GetDesc1](#)

Gets a description of the swap chain.

[GetDesc1](#)

Get an extended description of the output that includes color characteristics and connection type.

[GetDesc2](#)

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.2 description of an adapter or video card.

[GetDesc3](#)

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.6 description of an adapter or video card. This description includes information about ACG compatibility.

[GetDestSize](#)

Gets the size of the destination surface to use for the video processing blit operation.

[GetDevice](#)

Retrieves the device.

[GetDisplayModeList](#)

Gets the display modes that match the requested format and other input options.
(`IDXGIOutput.GetDisplayModeList`)

[GetDisplayModeList1](#)

Gets the display modes that match the requested format and other input options.
(`IDXGIOutput1.GetDisplayModeList1`)

[GetDisplaySurfaceData](#)

Gets a copy of the current display surface.

[GetDisplaySurfaceData1](#)

Copies the display surface (front buffer) to a user-provided resource.

[GetEvictionPriority](#)

Get the eviction priority.

[GetFrameDirtyRects](#)

Gets information about dirty rectangles for the current desktop frame.

[GetFrameLatencyWaitableObject](#)

Returns a waitable handle that signals when the DXGI adapter has finished presenting a new frame.

[GetFrameMoveRects](#)

Gets information about the moved rectangles for the current desktop frame.

[GetFramePointerShape](#)

Gets information about the new pointer shape for the current desktop frame.

[GetFrameStatistics](#)

Gets statistics about recently rendered frames.

[GetFrameStatistics](#)

Gets performance statistics about the last render frame.

[GetFrameStatisticsMedia](#)

Queries the system for a DXGI_FRAME_STATISTICS_MEDIA structure that indicates whether a custom refresh rate is currently approved by the system.

[GetFullscreenDesc](#)

Gets a description of a full-screen swap chain.

[GetFullscreenState](#)

Get the state associated with full-screen mode.

[GetGammaControl](#)

Gets the gamma control settings.

[GetGammaControlCapabilities](#)

Gets a description of the gamma-control capabilities.

[GetGPUThreadPriority](#)

Gets the GPU thread priority.

[GetHwnd](#)

Retrieves the underlying HWND for this swap-chain object.

[GetLastPresentCount](#)

Gets the number of times that IDXGISwapChain::Present or IDXGISwapChain1::Present1 has been called.

[GetMatrixTransform](#)

Gets the transform matrix that will be applied to a composition swap chain upon the next present.

[GetMaximumFrameLatency](#)

Gets the number of frames that the system is allowed to queue for rendering.

[GetMaximumFrameLatency](#)

Gets the number of frames that the swap chain is allowed to queue for rendering.

[GetMessage](#)

Gets a message from the message queue.

[GetMessageCountLimit](#)

Gets the maximum number of messages that can be added to the message queue.

[GetMuteDebugOutput](#)

Determines whether the debug output is turned on or off.

[GetNumMessagesAllowedByStorageFilter](#)

Gets the number of messages that a storage filter allowed to pass through.

[GetNumMessagesDeniedByStorageFilter](#)

Gets the number of messages that were denied passage through a storage filter.

[GetNumMessagesDiscardedByMessageCountLimit](#)

Gets the number of messages that were discarded due to the message count limit.

[GetNumStoredMessages](#)

Gets the number of messages currently stored in the message queue.

[GetNumStoredMessagesAllowedByRetrievalFilters](#)

Gets the number of messages that can pass through a retrieval filter.

GetParent
Gets the parent of the object.
GetPrivateData
Get a pointer to the object's data.
GetResource
Gets the parent resource and subresource index that support a subresource surface.
GetRestrictToOutput
Gets the output (the display monitor) to which you can restrict the contents of a present operation.
GetRetrievalFilter
Gets the retrieval filter at the top of the retrieval-filter stack.
GetRetrievalFilterStackSize
Gets the size of the retrieval-filter stack in bytes.
GetRotation
Gets the rotation of the back buffers for the swap chain.
GetSharedHandle
Gets the handle to a shared resource.
GetSharedResourceAdapterLuid
Identifies the adapter on which a shared resource object was created.
GetSourceRect
Gets the source region that is used for the swap chain.
GetSourceSize
Gets the source region used for the swap chain.
GetStorageFilter
Gets the storage filter at the top of the storage-filter stack.

<p>GetStorageFilterStackSize</p> <p>Gets the size of the storage-filter stack in bytes.</p>
<p>GetTargetRect</p> <p>Gets the rectangle that defines the target region for the video processing blit operation.</p>
<p>GetUsage</p> <p>Get the expected resource usage.</p>
<p>GetWindowAssociation</p> <p>Get the window through which the user controls the transition to and from full screen.</p>
<p>IsCurrent</p> <p>Informs an application of the possible need to re Enumerate adapters.</p>
<p>IsLeakTrackingEnabledForThread</p> <p>Gets a value indicating whether leak tracking is turned on for the current thread.</p>
<p>IsStereoEnabled</p> <p>Retrieves a Boolean value that indicates whether the operating system's stereoscopic 3D display behavior is enabled.</p>
<p>IsTemporaryMonoSupported</p> <p>Determines whether a swap chain supports "temporary mono."</p>
<p>IsWindowedStereoEnabled</p> <p>Determines whether to use stereo mode.</p>
<p>MakeWindowAssociation</p> <p>Allows DXGI to monitor an application's message queue for the alt-enter key sequence (which causes the application to switch from windowed to full screen or vice versa).</p>
<p>Map</p> <p>Get a pointer to the data contained in the surface, and deny GPU access to the surface.</p>

[MapDesktopSurface](#)

Provides the CPU with efficient access to a desktop image if that desktop image is already in system memory.

[OfferResources](#)

Allows the operating system to free the video memory of resources by discarding their content.
(IDXGIDevice2.OfferResources)

[OfferResources1](#)

Allows the operating system to free the video memory of resources, including both discarding the content and de-committing the memory.

[PopRetrievalFilter](#)

Pops a retrieval filter from the top of the retrieval-filter stack.

[PopStorageFilter](#)

Pops a storage filter from the top of the storage-filter stack.

[Present](#)

Presents a rendered image to the user.

[Present1](#)

Presents a frame on the display screen.

[PresentBuffer](#)

Presents a frame on the output adapter.

[PushCopyOfRetrievalFilter](#)

Pushes a copy of the retrieval filter that is currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

[PushCopyOfStorageFilter](#)

Pushes a copy of the storage filter that is currently on the top of the storage-filter stack onto the storage-filter stack.

[PushDenyAllRetrievalFilter](#)

Pushes a deny-all retrieval filter onto the retrieval-filter stack.

PushDenyAllStorageFilter
Pushes a deny-all storage filter onto the storage-filter stack.
PushEmptyRetrievalFilter
Pushes an empty retrieval filter onto the retrieval-filter stack.
PushEmptyStorageFilter
Pushes an empty storage filter onto the storage-filter stack.
PushRetrievalFilter
Pushes a retrieval filter onto the retrieval-filter stack.
PushStorageFilter
Pushes a storage filter onto the storage-filter stack.
QueryResourceResidency
Gets the residency status of an array of resources.
QueryVideoMemoryInfo
This method informs the process of the current budget and process usage.
ReclaimResources
Restores access to resources that were previously offered by calling IDXGIDevice2::OfferResources.
ReclaimResources1
Restores access to resources that were previously offered by calling IDXGIDevice4::OfferResources1.
RegisterAdaptersChangedEvent
Registers to receive notification of changes whenever the adapter enumeration state changes.
RegisterHardwareContentProtectionTeardownStatusEvent
Registers to receive notification of hardware content protection teardown events.
RegisterOcclusionStatusEvent
Registers to receive notification of changes in occlusion status by using event signaling.

[RegisterOcclusionStatusWindow](#)

Registers an application window to receive notification messages of changes of occlusion status.

[RegisterStereoStatusEvent](#)

Registers to receive notification of changes in stereo status by using event signaling.

[RegisterStereoStatusWindow](#)

Registers an application window to receive notification messages of changes of stereo status.

[RegisterVideoMemoryBudgetChangeNotificationEvent](#)

This method establishes a correlation between a CPU synchronization object and the budget change event.

[ReleaseDC](#)

Releases the GDI device context (DC) that is associated with the current surface and allows you to use Direct3D to render.

[ReleaseFrame](#)

Indicates that the application finished processing the frame.

[ReleaseOwnership](#)

Releases ownership of the output.

[ReleaseSync](#)

Using a key, releases exclusive rendering access to a shared resource.

[ReportLiveObjects](#)

Reports info about the lifetime of an object or objects.

[ResizeBuffers](#)

Changes the swap chain's back buffer size, format, and number of buffers. This should be called when the application window is resized.

[ResizeBuffers1](#)

Changes the swap chain's back buffer size, format, and number of buffers, where the swap chain was created using a D3D12 command queue as an input device. This should be called when the application window is resized.

ResizeTarget
Resizes the output target.
SetBackgroundColor
Changes the background color of the swap chain.
SetBreakOnCategory
Sets a message category to break on when a message with that category passes through the storage filter.
SetBreakOnID
Sets a message identifier to break on when a message with that identifier passes through the storage filter.
SetBreakOnSeverity
Sets a message severity level to break on when a message with that severity level passes through the storage filter.
SetColorSpace
Sets the color space used by the swap chain. (<code>IDXGIDecodeSwapChain.SetColorSpace</code>)
SetColorSpace1
Sets the color space used by the swap chain. (<code>IDXGISwapChain3SetColorSpace1</code>)
SetDestSize
Sets the size of the destination surface to use for the video processing blit operation.
SetDisplaySurface
Changes the display mode.
SetEvictionPriority
Set the priority for evicting the resource from memory.
SetFullscreenState
Sets the display state to windowed or full screen.

[SetGammaControl](#)

Sets the gamma controls.

[SetGPUThreadPriority](#)

Sets the GPU thread priority.

[SetHDRMetaData](#)

This method sets High Dynamic Range (HDR) and Wide Color Gamut (WCG) header metadata.

[SetMatrixTransform](#)

Sets the transform matrix that will be applied to a composition swap chain upon the next present.

[SetMaximumFrameLatency](#)

Sets the number of frames that the system is allowed to queue for rendering.

[SetMaximumFrameLatency](#)

Sets the number of frames that the swap chain is allowed to queue for rendering.

[SetMessageCountLimit](#)

Sets the maximum number of messages that can be added to the message queue.

[SetMuteDebugOutput](#)

Turns the debug output on or off.

[SetPresentDuration](#)

Requests a custom presentation duration (custom refresh rate).

[SetPrivateData](#)

Sets application-defined data to the object and associates that data with a GUID.

[SetPrivateDataInterface](#)

Set an interface in the object's private data.

[SetRotation](#)

Sets the rotation of the back buffers for the swap chain.

SetSourceRect
Sets the rectangle that defines the source region for the video processing blit operation.
SetSourceSize
Sets the source region to be used for the swap chain.
SetStereoEnabled
Set a Boolean value to either enable or disable the operating system's stereoscopic 3D display behavior.
SetTargetRect
Sets the rectangle that defines the target region for the video processing blit operation.
SetVideoMemoryReservation
This method sends the minimum required physical memory for an application, to the OS.
SupportsOverlays
Queries an adapter output for multiplane overlay support.
TakeOwnership
Takes ownership of an output.
Trim
Trims the graphics memory allocated by the IDXGIDevice3 DXGI device on the app's behalf.
Unmap
Invalidate the pointer to the surface retrieved by IDXGISurface::Map and re-enable GPU access to the resource.
UnMapDesktopSurface
Invalidates the pointer to the desktop image that was retrieved by using IDXGIOutputDuplication::MapDesktopSurface.
UnregisterAdaptersChangedEvent
Registers an event to stop receiving notifications when the adapter enumeration state changes.

[UnregisterHardwareContentProtectionTeardownStatus](#)

Unregisters an event to stop it from receiving notification of hardware content protection teardown events.

[UnregisterOcclusionStatus](#)

Unregisters a window or an event to stop it from receiving notification when occlusion status changes.

[UnregisterStereoStatus](#)

Unregisters a window or an event to stop it from receiving notification when stereo status changes.

[UnregisterVideoMemoryBudgetChangeNotification](#)

This method stops notifying a CPU synchronization object whenever a budget change occurs. An application may switch back to polling the information regularly.

[WaitForVBlank](#)

Halt a thread until the next vertical blank occurs.

Interfaces

[IDXGIAdapter](#)

The IDXGIAdapter interface represents a display subsystem (including one or more GPUs, DACs and video memory).

[IDXGIAdapter1](#)

The IDXGIAdapter1 interface represents a display sub-system (including one or more GPU's, DACs and video memory).

[IDXGIAdapter2](#)

The IDXGIAdapter2 interface represents a display subsystem, which includes one or more GPUs, DACs, and video memory.

[IDXGIAdapter3](#)

This interface adds some memory residency methods, for budgeting and reserving physical memory.

[IDXGIAdapter4](#)

This interface represents a display subsystem, and extends this family of interfaces to expose a method to check for an adapter's compatibility with Arbitrary Code Guard (ACG).

[IDXGIDebug](#)

This interface controls debug settings, and can only be used if the debug layer is turned on.

[IDXGIDebug1](#)

Controls debug settings for Microsoft DirectX Graphics Infrastructure (DXGI). You can use the IDXGIDebug1 interface in Windows Store apps.

[IDXGIDeclareSwapChain](#)

Represents a swap chain that is used by desktop media apps to decode video data and show it on a DirectComposition surface.

[IDXGIDevice](#)

An IDXGIDevice interface implements a derived class for DXGI objects that produce image data.

[IDXGIDevice1](#)

An IDXGIDevice1 interface implements a derived class for DXGI objects that produce image data.

[IDXGIDevice2](#)

The IDXGIDevice2 interface implements a derived class for DXGI objects that produce image data. The interface exposes methods to block CPU processing until the GPU completes processing, and to offer resources to the operating system.

[IDXGIDevice3](#)

The IDXGIDevice3 interface implements a derived class for DXGI objects that produce image data. The interface exposes a method to trim graphics memory usage by the DXGI device.

[IDXGIDevice4](#)

This interface provides updated methods to offer and reclaim resources.

[IDXGIDeviceSubObject](#)

Inherited from objects that are tied to the device so that they can retrieve a pointer to it.

[IDXGIDisplayControl](#)

The IDXGIDisplayControl interface exposes methods to indicate user preference for the operating system's stereoscopic 3D display behavior and to set stereoscopic 3D display status to enable or disable.

[IDXGIFactory](#)

An IDXGIFactory interface implements methods for generating DXGI objects (which handle full screen transitions).

[IDXGIFactory1](#)

The IDXGIFactory1 interface implements methods for generating DXGI objects.

[IDXGIFactory2](#)

The IDXGIFactory2 interface includes methods to create a newer version swap chain with more features than IDXGISwapChain and to monitor stereoscopic 3D capabilities.

[IDXGIFactory3](#)

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects. (IDXGIFactory3)

[IDXGIFactory4](#)

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects. (IDXGIFactory4)

[IDXGIFactory5](#)

This interface enables a single method to support variable refresh rate displays.

[IDXGIFactory6](#)

This interface enables a single method that enumerates graphics adapters based on a given GPU preference.

[IDXGIFactory7](#)

This interface enables registration for notifications to detect adapter enumeration state changes.

[IDXGIFactoryMedia](#)

Creates swap chains for desktop media apps that use DirectComposition surfaces to decode and display video.

[IDXGIFInfoQueue](#)

This interface controls the debug information queue, and can only be used if the debug layer is turned on.

[IDXGIKeyedMutex](#)

Represents a keyed mutex, which allows exclusive access to a shared resource that is used by multiple devices.

[IDXGIOObject](#)

An IDXGIOObject interface is a base interface for all DXGI objects; IDXGIOObject supports associating caller-defined (private data) with an object and retrieval of an interface to the parent object.

[IDXGIOOutput](#)

An IDXGIOOutput interface represents an adapter output (such as a monitor).

[IDXGIOOutput1](#)

An IDXGIOOutput1 interface represents an adapter output (such as a monitor).

[IDXGIOOutput2](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput2 interface exposes a method to check for multiplane overlay support on the primary output adapter.

[IDXGIOOutput3](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput3 interface exposes a method to check for overlay support.

[IDXGIOOutput4](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput4 interface exposes a method to check for overlay color space support.

[IDXGIOOutput5](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput5 interface exposes a single method to specify a list of supported formats for fullscreen surfaces.

[IDXGIOOutput6](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput6 interface exposes methods to provide specific monitor capabilities.

[IDXGIOutputDuplication](#)

The IDXGIOutputDuplication interface accesses and manipulates the duplicated desktop image.

[IDXGIResource](#)

An IDXGIResource interface allows resource sharing and identifies the memory that a resource resides in.

[IDXGIResource1](#)

An IDXGIResource1 interface extends the IDXGIResource interface by adding support for creating a subresource surface object and for creating a handle to a shared resource.

[IDXGISurface](#)

The IDXGISurface interface implements methods for image-data objects.

[IDXGISurface1](#)

The IDXGISurface1 interface extends the IDXGISurface by adding support for using Windows Graphics Device Interface (GDI) to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface.

[IDXGISurface2](#)

The IDXGISurface2 interface extends the IDXGISurface1 interface by adding support for subresource surfaces and getting a handle to a shared resource.

[IDXGISwapChain](#)

An IDXGISwapChain interface implements one or more surfaces for storing rendered data before presenting it to an output.

[IDXGISwapChain1](#)

Provides presentation capabilities that are enhanced from IDXGISwapChain. These presentation capabilities consist of specifying dirty rectangles and scroll rectangle to optimize the presentation.

[IDXGISwapChain2](#)

Extends IDXGISwapChain1 with methods to support swap back buffer scaling and lower-latency swap chains.

[IDXGISwapChain3](#)

Extends IDXGISwapChain2 with methods to support getting the index of the swap chain's current back buffer and support for color space.

[IDXGISwapChain4](#)

This interface exposes a single method for setting video metadata.

[IDXGISwapChainMedia](#)

This swap chain interface allows desktop media applications to request a seamless change to a specific refresh rate.

Structures

[DXGI_ADAPTER_DESC](#)

Describes an adapter (or video card) by using DXGI 1.0.

[DXGI_ADAPTER_DESC1](#)

Describes an adapter (or video card) using DXGI 1.1.

[DXGI_ADAPTER_DESC2](#)

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.2.

[DXGI_ADAPTER_DESC3](#)

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.6.

[DXGI_DECODE_SWAP_CHAIN_DESC](#)

Used with IDXGIFactoryMedia::CreateDecodeSwapChainForCompositionSurfaceHandle to describe a decode swap chain.

[DXGI_DISPLAY_COLOR_SPACE](#)

Don't use this structure; it is not supported and it will be removed from the header in a future release.

[DXGI_FRAME_STATISTICS](#)

Describes timing and presentation statistics for a frame.

DXGI_FRAME_STATISTICS_MEDIA	Used to verify system approval for the app's custom present duration (custom refresh rate).
DXGI_HDR_METADATA_HDR10	Describes the metadata for HDR10, used when video is compressed using High Efficiency Video Coding (HEVC).
DXGI_INFO_QUEUE_FILTER	Describes a debug message filter, which contains lists of message types to allow and deny.
DXGI_INFO_QUEUE_FILTER_DESC	Describes the types of messages to allow or deny to pass through a filter.
DXGI_INFO_QUEUE_MESSAGE	Describes a debug message in the information queue.
DXGI_MAPPED_RECT	Describes a mapped rectangle that is used to access a surface.
DXGI_MATRIX_3X2_F	Represents a 3x2 matrix. Used with GetMatrixTransform and SetMatrixTransform to indicate the scaling and translation transform for SwapChainPanel swap chains.
DXGI_MODE_DESC1	Describes a display mode and whether the display mode supports stereo.
DXGI_OUTDUPL_DESC	The DXGI_OUTDUPL_DESC structure describes the dimension of the output and the surface that contains the desktop image. The format of the desktop image is always DXGI_FORMAT_B8G8R8A8_UNORM.
DXGI_OUTDUPL_FRAME_INFO	The DXGI_OUTDUPL_FRAME_INFO structure describes the current desktop image.
DXGI_OUTDUPL_MOVE_RECT	The DXGI_OUTDUPL_MOVE_RECT structure describes the movement of a rectangle.

[DXGI_OUTDUPL_POINTER_POSITION](#)

The DXGI_OUTDUPL_POINTER_POSITION structure describes the position of the hardware cursor.

[DXGI_OUTDUPL_POINTER_SHAPE_INFO](#)

The DXGI_OUTDUPL_POINTER_SHAPE_INFO structure describes information about the cursor shape.

[DXGI_OUTPUT_DESC](#)

Describes an output or physical connection between the adapter (video card) and a device.

[DXGI_OUTPUT_DESC1](#)

Describes an output or physical connection between the adapter (video card) and a device, including additional information about color capabilities and connection type.

[DXGI_PRESENT_PARAMETERS](#)

Describes information about present that helps the operating system optimize presentation.

[DXGI_QUERY_VIDEO_MEMORY_INFO](#)

Describes the current video memory budgeting parameters.

[DXGI_RATIONAL](#)

Represents a rational number.

[DXGI_SAMPLE_DESC](#)

Describes multi-sampling parameters for a resource.

[DXGI_SHARED_RESOURCE](#)

Represents a handle to a shared resource.

[DXGI_SURFACE_DESC](#)

Describes a surface.

[DXGI_SWAP_CHAIN_DESC](#)

Describes a swap chain. (DXGI_SWAP_CHAIN_DESC)

[DXGI_SWAP_CHAIN_DESC1](#)

Describes a swap chain. (DXGI_SWAP_CHAIN_DESC1)

[DXGI_SWAP_CHAIN_FULLSCREEN_DESC](#)

Describes full-screen mode for a swap chain.

[LUID](#)

Describes a local identifier for an adapter. (LUID)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgi.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi.h contains the following programming interfaces:

Interfaces

[IDXGIAdapter](#)

The IDXGIAdapter interface represents a display subsystem (including one or more GPUs, DACs and video memory).

[IDXGIAdapter1](#)

The IDXGIAdapter1 interface represents a display sub-system (including one or more GPU's, DACs and video memory).

[IDXGIDevice](#)

An IDXGIDevice interface implements a derived class for DXGI objects that produce image data.

[IDXGIDevice1](#)

An IDXGIDevice1 interface implements a derived class for DXGI objects that produce image data.

[IDXGIDeviceSubObject](#)

Inherited from objects that are tied to the device so that they can retrieve a pointer to it.

[IDXGIFactory](#)

An IDXGIFactory interface implements methods for generating DXGI objects (which handle full screen transitions).

[IDXGIFactory1](#)

The IDXGIFactory1 interface implements methods for generating DXGI objects.

[IDXGIMutex](#)

Represents a keyed mutex, which allows exclusive access to a shared resource that is used by multiple devices.

[IDXGIOBJECT](#)

An IDXGIOBJECT interface is a base interface for all DXGI objects; IDXGIOBJECT supports associating caller-defined (private data) with an object and retrieval of an interface to the parent object.

[IDXGIOUTPUT](#)

An IDXGIOUTPUT interface represents an adapter output (such as a monitor).

[IDXGIRESOURCE](#)

An IDXGIRESOURCE interface allows resource sharing and identifies the memory that a resource resides in.

[IDXGISURFACE](#)

The IDXGISURFACE interface implements methods for image-data objects.

[IDXGISURFACE1](#)

The IDXGISURFACE1 interface extends the IDXGISURFACE by adding support for using Windows Graphics Device Interface (GDI) to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface.

[IDXGISWAPCHAIN](#)

An IDXGISWAPCHAIN interface implements one or more surfaces for storing rendered data before presenting it to an output.

Functions

[CreateDXGIFactory](#)

Creates a DXGI 1.0 factory that you can use to generate other DXGI objects.

[CreateDXGIFactory1](#)

Creates a DXGI 1.1 factory that you can use to generate other DXGI objects.

Structures

DXGI_ADAPTER_DESC	Describes an adapter (or video card) by using DXGI 1.0.
DXGI_ADAPTER_DESC1	Describes an adapter (or video card) using DXGI 1.1.
DXGI_DISPLAY_COLOR_SPACE	Don't use this structure; it is not supported and it will be removed from the header in a future release.
DXGI_FRAME_STATISTICS	Describes timing and presentation statistics for a frame.
DXGI_MAPPED_RECT	Describes a mapped rectangle that is used to access a surface.
DXGI_OUTPUT_DESC	Describes an output or physical connection between the adapter (video card) and a device.
DXGI_SHARED_RESOURCE	Represents a handle to a shared resource.
DXGI_SURFACE_DESC	Describes a surface.
DXGI_SWAP_CHAIN_DESC	Describes a swap chain. (DXGI_SWAP_CHAIN_DESC)

Enumerations

DXGI_ADAPTER_FLAG	Identifies the type of DXGI adapter. (DXGI_ADAPTER_FLAG)
-----------------------------------	--

[DXGI_RESIDENCY](#)

Flags indicating the memory location of a resource.

[DXGI_SWAP_CHAIN_FLAG](#)

Options for swap-chain behavior.

[DXGI_SWAP_EFFECT](#)

Options for handling pixels in a display surface after calling IDXGISwapChain1::Present1.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

CreateDXGIFactory function (dxgi.h)

Article10/13/2021

Creates a DXGI 1.0 factory that you can use to generate other DXGI objects.

Syntax

C++

```
HRESULT CreateDXGIFactory(
    REFIID riid,
    [out] void    **ppFactory
);
```

Parameters

riid

Type: [REFIID](#)

The globally unique identifier (GUID) of the [IDXGIFactory](#) object referenced by the *ppFactory* parameter.

[out] ppFactory

Type: [void**](#)

Address of a pointer to an [IDXGIFactory](#) object.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the following [DXGI_ERROR](#).

Remarks

Use a DXGI factory to generate objects that [enumerate adapters](#), [create swap chains](#), and [associate a window](#) with the alt+enter key sequence for toggling to and from the fullscreen display mode.

If the **CreateDXGIFactory** function succeeds, the reference count on the **IDXGIFactory** interface is incremented. To avoid a memory leak, when you finish using the interface, call the **IDXGIFactory::Release** method to release the interface.

Note Do not mix the use of DXGI 1.0 (**IDXGIFactory**) and DXGI 1.1 (**IDXGIFactory1**) in an application. Use **IDXGIFactory** or **IDXGIFactory1**, but not both in an application.

Note **CreateDXGIFactory** fails if your app's **DllMain** function calls it. For more info about how DXGI responds from **DllMain**, see **DXGI Responses from DllMain**.

Note Starting with Windows 8, all DXGI factories (regardless if they were created with **CreateDXGIFactory** or **CreateDXGIFactory1**) enumerate adapters identically. The enumeration order of adapters, which you retrieve with **IDXGIFactory::EnumAdapters** or **IDXGIFactory1::EnumAdapters1**, is as follows:

- Adapter with the output on which the desktop primary is displayed. This adapter corresponds with an index of zero.
- Adapters with outputs.
- Adapters without outputs.

The **CreateDXGIFactory** function does not exist for Windows Store apps. Instead, Windows Store apps use the **CreateDXGIFactory1** function.

Examples

Creating a DXGI 1.0 Factory

The following code example demonstrates how to create a DXGI 1.0 factory. This example uses the `_uuidof()` intrinsic to obtain the REFIID, or GUID, of the **IDXGIFactory** interface.

```
IDXGIFactory * pFactory;  
HRESULT hr = CreateDXGIFactory(_uuidof(IDXGIFactory), (void**)(&pFactory)
```

);

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib
DLL	DXGI.dll

See also

[DXGI Functions](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

CreateDXGIFactory1 function (dxgi.h)

Article 08/23/2022

Creates a DXGI 1.1 factory that you can use to generate other DXGI objects.

Syntax

C++

```
HRESULT CreateDXGIFactory1(
    REFIID riid,
    [out] void    **ppFactory
);
```

Parameters

riid

Type: [REFIID](#)

The globally unique identifier (GUID) of the [IDXGIFactory1](#) object referenced by the *ppFactory* parameter.

[out] ppFactory

Type: [void**](#)

Address of a pointer to an [IDXGIFactory1](#) object.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Use a DXGI 1.1 factory to generate objects that [enumerate adapters](#), [create swap chains](#), and [associate a window](#) with the alt+enter key sequence for toggling to and from the full-screen display mode.

If the [CreateDXGIFactory1](#) function succeeds, the reference count on the [IDXGIFactory1](#) interface is incremented. To avoid a memory leak, when you finish using the interface, call the [IDXGIFactory1::Release](#) method to release the interface.

This entry point is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Note Do not mix the use of DXGI 1.0 ([IDXGIFactory](#)) and DXGI 1.1 ([IDXGIFactory1](#)) in an application. Use [IDXGIFactory](#) or [IDXGIFactory1](#), but not both in an application.

Note [CreateDXGIFactory1](#) fails if your app's [DII Main](#) function calls it. For more info about how DXGI responds from [DII Main](#), see [DXGI Responses from DLLMain](#).

Note Starting with Windows 8, all DXGI factories (regardless if they were created with [CreateDXGIFactory](#) or [CreateDXGIFactory1](#)) enumerate adapters identically. The enumeration order of adapters, which you retrieve with [IDXGIFactory::EnumAdapters](#) or [IDXGIFactory1::EnumAdapters1](#), is as follows:

- Adapter with the output on which the desktop primary is displayed. This adapter corresponds with an index of zero.
- Adapters with outputs.
- Adapters without outputs.

Examples

Creating a DXGI 1.1 Factory

The following code example demonstrates how to create a DXGI 1.1 factory. This example uses the `_uuidof()` intrinsic to obtain the REFIID, or GUID, of the [IDXGIFactory1](#) interface.

```
IDXGIFactory1 * pFactory;  
HRESULT hr = CreateDXGIFactory1(__uuidof(IDXGIFactory1), (void**)(&pFactory)  
);
```

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib
DLL	Dxgi.dll

See also

[DXGI Functions](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_ADAPTER_DESC structure (dxgi.h)

Article09/01/2022

Describes an adapter (or video card) by using DXGI 1.0.

Syntax

C++

```
typedef struct DXGI_ADAPTER_DESC {
    WCHAR Description[128];
    UINT VendorId;
    UINT DeviceId;
    UINT SubSysId;
    UINT Revision;
    SIZE_T DedicatedVideoMemory;
    SIZE_T DedicatedSystemMemory;
    SIZE_T SharedSystemMemory;
    LUID AdapterLuid;
} DXGI_ADAPTER_DESC;
```

Members

Description[128]

Type: **WCHAR[128]**

A string that contains the adapter description. On [feature level](#) 9 graphics hardware, [GetDesc](#) returns "Software Adapter" for the description string.

VendorId

Type: **UINT**

The PCI ID of the hardware vendor. On [feature level](#) 9 graphics hardware, [GetDesc](#) returns zeros for the PCI ID of the hardware vendor.

DeviceId

Type: **UINT**

The PCI ID of the hardware device. On [feature level](#) 9 graphics hardware, [GetDesc](#) returns zeros for the PCI ID of the hardware device.

SubSysId

Type: [UINT](#)

The PCI ID of the sub system. On [feature level](#) 9 graphics hardware, [GetDesc](#) returns zeros for the PCI ID of the sub system.

Revision

Type: [UINT](#)

The PCI ID of the revision number of the adapter. On [feature level](#) 9 graphics hardware, [GetDesc](#) returns zeros for the PCI ID of the revision number of the adapter.

DedicatedVideoMemory

Type: [SIZE_T](#)

The number of bytes of dedicated video memory that are not shared with the CPU.

DedicatedSystemMemory

Type: [SIZE_T](#)

The number of bytes of dedicated system memory that are not shared with the CPU. This memory is allocated from available system memory at boot time.

SharedSystemMemory

Type: [SIZE_T](#)

The number of bytes of shared system memory. This is the maximum value of system memory that may be consumed by the adapter during operation. Any incidental memory consumed by the driver as it manages and uses video memory is additional.

AdapterLuid

Type: [LUID](#)

A unique value that identifies the adapter. See [LUID](#) for a definition of the structure. LUID is defined in dxgi.h.

Remarks

The **DXGI_ADAPTER_DESC** structure provides a description of an adapter. This structure is initialized by using the [IDXGIAdapter::GetDesc](#) method.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_ADAPTER_DESC1 structure (dxgi.h)

Article 09/01/2022

Describes an adapter (or video card) using DXGI 1.1.

Syntax

C++

```
typedef struct DXGI_ADAPTER_DESC1 {
    WCHAR Description[128];
    UINT VendorId;
    UINT DeviceId;
    UINT SubSysId;
    UINT Revision;
    SIZE_T DedicatedVideoMemory;
    SIZE_T DedicatedSystemMemory;
    SIZE_T SharedSystemMemory;
    LUID AdapterLuid;
    UINT Flags;
} DXGI_ADAPTER_DESC1;
```

Members

Description[128]

Type: **WCHAR[128]**

A string that contains the adapter description. On [feature level](#) 9 graphics hardware, [GetDesc1](#) returns “Software Adapter” for the description string.

VendorId

Type: **UINT**

The PCI ID of the hardware vendor. On [feature level](#) 9 graphics hardware, [GetDesc1](#) returns zeros for the PCI ID of the hardware vendor.

DeviceId

Type: **UINT**

The PCI ID of the hardware device. On [feature level](#) 9 graphics hardware, [GetDesc1](#) returns zeros for the PCI ID of the hardware device.

SubSysId

Type: **UINT**

The PCI ID of the sub system. On [feature level](#) 9 graphics hardware, [GetDesc1](#) returns zeros for the PCI ID of the sub system.

Revision

Type: **UINT**

The PCI ID of the revision number of the adapter. On [feature level](#) 9 graphics hardware, [GetDesc1](#) returns zeros for the PCI ID of the revision number of the adapter.

DedicatedVideoMemory

Type: **SIZE_T**

The number of bytes of dedicated video memory that are not shared with the CPU.

DedicatedSystemMemory

Type: **SIZE_T**

The number of bytes of dedicated system memory that are not shared with the CPU. This memory is allocated from available system memory at boot time.

SharedSystemMemory

Type: **SIZE_T**

The number of bytes of shared system memory. This is the maximum value of system memory that may be consumed by the adapter during operation. Any incidental memory consumed by the driver as it manages and uses video memory is additional.

AdapterLuid

Type: **LUID**

A unique value that identifies the adapter. See [LUID](#) for a definition of the structure. LUID is defined in dxgi.h.

Flags

Type: **UINT**

A value of the [DXGI_ADAPTER_FLAG](#) enumerated type that describes the adapter type. The [DXGI_ADAPTER_FLAG_REMOTE](#) flag is reserved.

Remarks

The [DXGI_ADAPTER_DESC1](#) structure provides a DXGI 1.1 description of an adapter. This structure is initialized by using the [IDXGIAdapter1::GetDesc1](#) method.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Structures](#)

[IDXGIAdapter1::GetDesc1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_ADAPTER_FLAG enumeration (dxgi.h)

Article 07/27/2022

Identifies the type of DXGI adapter.

Syntax

C++

```
typedef enum DXGI_ADAPTER_FLAG {
    DXGI_ADAPTER_FLAG_NONE = 0,
    DXGI_ADAPTER_FLAG_REMOTE = 1,
    DXGI_ADAPTER_FLAG_SOFTWARE = 2,
    DXGI_ADAPTER_FLAG_FORCE_DWORD = 0xffffffff
};
```

Constants

DXGI_ADAPTER_FLAG_NONE

Value: 0

Specifies no flags.

DXGI_ADAPTER_FLAG_REMOTE

Value: 1

Value always set to 0. This flag is reserved.

DXGI_ADAPTER_FLAG_SOFTWARE

Value: 2

Specifies a software adapter. For more info about this flag, see [new info in Windows 8 about enumerating adapters](#).

Direct3D 11: This enumeration value is supported starting with Windows 8.

DXGI_ADAPTER_FLAG_FORCE_DWORD

Value: 0xffffffff

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

Remarks

The `DXGI_ADAPTER_FLAG` enumerated type is used by the `Flags` member of the `DXGI_ADAPTER_DESC1` or `DXGI_ADAPTER_DESC2` structure to identify the type of DXGI adapter.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes	 No
---	--

[Get help at Microsoft Q&A](#)

DXGI_DISPLAY_COLOR_SPACE structure (dxgi.h)

Article09/01/2022

Don't use this structure; it is not supported and it will be removed from the header in a future release.

Syntax

C++

```
typedef struct DXGI_DISPLAY_COLOR_SPACE {
    FLOAT PrimaryCoordinates[8][2];
    FLOAT WhitePoints[16][2];
} DXGI_DISPLAY_COLOR_SPACE;
```

Members

`PrimaryCoordinates[8]`

The primary coordinates, as an 8 by 2 array of FLOAT values.

`WhitePoints[16]`

The white points, as a 16 by 2 array of FLOAT values.

Requirements

Header

dxgi.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_FRAME_STATISTICS structure (dxgi.h)

Article04/02/2021

Describes timing and presentation statistics for a frame.

Syntax

C++

```
typedef struct DXGI_FRAME_STATISTICS {
    UINT PresentCount;
    UINT PresentRefreshCount;
    UINT SyncRefreshCount;
    LARGE_INTEGER SyncQPCTime;
    LARGE_INTEGER SyncGPUTime;
} DXGI_FRAME_STATISTICS;
```

Members

PresentCount

Type: [UINT](#)

A value that represents the running total count of times that an image was presented to the monitor since the computer booted.

Note The number of times that an image was presented to the monitor is not necessarily the same as the number of times that you called [IDXGISwapChain::Present](#) or [IDXGISwapChain1::Present1](#).

PresentRefreshCount

Type: [UINT](#)

A value that represents the running total count of v-blanks at which the last image was presented to the monitor and that have happened since the computer booted (for windowed mode, since the swap chain was created).

SyncRefreshCount

Type: **UINT**

A value that represents the running total count of v-blanks when the scheduler last sampled the machine time by calling [QueryPerformanceCounter](#) and that have happened since the computer booted (for windowed mode, since the swap chain was created).

SyncQPCTime

Type: **LARGE_INTEGER**

A value that represents the high-resolution performance counter timer. This value is the same as the value returned by the [QueryPerformanceCounter](#) function.

SyncGPUTime

Type: **LARGE_INTEGER**

Reserved. Always returns 0.

Remarks

You initialize the **DXGI_FRAME_STATISTICS** structure with the [IDXGIOutput::GetFrameStatistics](#) or [IDXGISwapChain::GetFrameStatistics](#) method.

You can only use [IDXGISwapChain::GetFrameStatistics](#) for swap chains that either use the flip presentation model or draw in full-screen mode. You set the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value in the **SwapEffect** member of the **DXGI_SWAP_CHAIN_DESC1** structure to specify that the swap chain uses the flip presentation model.

The values in the **PresentCount** and **PresentRefreshCount** members indicate information about when a frame was presented on the display screen. You can use these values to determine whether a glitch occurred. The values in the **SyncRefreshCount** and **SyncQPCTime** members indicate timing information that you can use for audio and video synchronization or very precise animation. If the swap chain draws in full-screen mode, these values are based on when the computer booted. If the swap chain draws in windowed mode, these values are based on when the swap chain is created.

Requirements

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_MAPPED_RECT structure (dxgi.h)

Article 04/02/2021

Describes a mapped rectangle that is used to access a surface.

Syntax

C++

```
typedef struct DXGI_MAPPED_RECT {
    INT Pitch;
    BYTE *pBits;
} DXGI_MAPPED_RECT;
```

Members

Pitch

Type: [INT](#)

A value that describes the width, in bytes, of the surface.

pBits

Type: [BYTE*](#)

A pointer to the image buffer of the surface.

Remarks

The `DXGI_MAPPED_RECT` structure is initialized by the [IDXGISurface::Map](#) method.

Requirements

Header

dxgi.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTPUT_DESC structure (dxgi.h)

Article 09/01/2022

Describes an output or physical connection between the adapter (video card) and a device.

Syntax

C++

```
typedef struct DXGI_OUTPUT_DESC {
    WCHAR             DeviceName[32];
    RECT              DesktopCoordinates;
    BOOL              AttachedToDesktop;
    DXGI_MODE_ROTATION Rotation;
    HMONITOR          Monitor;
} DXGI_OUTPUT_DESC;
```

Members

DeviceName[32]

Type: **WCHAR[32]**

A string that contains the name of the output device.

DesktopCoordinates

Type: **RECT**

A **RECT** structure containing the bounds of the output in desktop coordinates. Desktop coordinates depend on the dots per inch (DPI) of the desktop. For info about writing DPI-aware Win32 apps, see [High DPI](#).

AttachedToDesktop

Type: **BOOL**

True if the output is attached to the desktop; otherwise, false.

Rotation

Type: **DXGI_MODE_ROTATION**

A member of the [DXGI_MODE_ROTATION](#) enumerated type describing on how an image is rotated by the output.

Monitor

Type: [HMONITOR](#)

An [HMONITOR](#) handle that represents the display monitor. For more information, see [HMONITOR and the Device Context](#).

Remarks

The [DXGI_OUTPUT_DESC](#) structure is initialized by the [IDXGIOutput::GetDesc](#) method.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_RESIDENCY enumeration (dxgi.h)

Article01/31/2022

Flags indicating the memory location of a resource.

Syntax

C++

```
typedef enum DXGI_RESIDENCY {
    DXGI_RESIDENCY_FULLY_RESIDENT = 1,
    DXGI_RESIDENCY_RESIDENT_IN_SHARED_MEMORY = 2,
    DXGI_RESIDENCY_EVICTED_TO_DISK = 3
} ;
```

Constants

`DXGI_RESIDENCY_FULLY_RESIDENT`

Value: 1

The resource is located in video memory.

`DXGI_RESIDENCY_RESIDENT_IN_SHARED_MEMORY`

Value: 2

At least some of the resource is located in CPU memory.

`DXGI_RESIDENCY_EVICTED_TO_DISK`

Value: 3

At least some of the resource has been paged out to the hard drive.

Remarks

This enum is used by [QueryResourceResidency](#).

Requirements

Header

dxgi.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SHARED_RESOURCE structure (dxgi.h)

Article04/02/2021

Represents a handle to a shared resource.

Syntax

C++

```
typedef struct DXGI_SHARED_RESOURCE {
    HANDLE Handle;
} DXGI_SHARED_RESOURCE;
```

Members

Handle

Type: [HANDLE](#)

A handle to a shared resource.

Remarks

To create a shared surface, pass a shared-resource handle into the [IDXGIDevice::CreateSurface](#) method.

Requirements

Header

dxgi.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SURFACE_DESC structure (dxgi.h)

Article 04/02/2021

Describes a surface.

Syntax

C++

```
typedef struct DXGI_SURFACE_DESC {  
    UINT           Width;  
    UINT           Height;  
    DXGI_FORMAT    Format;  
    DXGI_SAMPLE_DESC SampleDesc;  
} DXGI_SURFACE_DESC;
```

Members

Width

Type: **UINT**

A value describing the surface width.

Height

Type: **UINT**

A value describing the surface height.

Format

Type: **DXGI_FORMAT**

A member of the **DXGI_FORMAT** enumerated type that describes the surface format.

SampleDesc

Type: **DXGI_SAMPLE_DESC**

A member of the **DXGI_SAMPLE_DESC** structure that describes multi-sampling parameters for the surface.

Remarks

This structure is used by the [GetDesc](#) and [CreateSurface](#) methods.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SWAP_CHAIN_DESC structure (dxgi.h)

Article 07/27/2022

Describes a swap chain.

Syntax

C++

```
typedef struct DXGI_SWAP_CHAIN_DESC {
    DXGI_MODE_DESC    BufferDesc;
    DXGI_SAMPLE_DESC  SampleDesc;
    DXGI_USAGE        BufferUsage;
    UINT              BufferCount;
    HWND              OutputWindow;
    BOOL              Windowed;
    DXGI_SWAP_EFFECT  SwapEffect;
    UINT              Flags;
} DXGI_SWAP_CHAIN_DESC;
```

Members

BufferDesc

Type: [DXGI_MODE_DESC](#)

A [DXGI_MODE_DESC](#) structure that describes the backbuffer display mode.

SampleDesc

Type: [DXGI_SAMPLE_DESC](#)

A [DXGI_SAMPLE_DESC](#) structure that describes multi-sampling parameters.

BufferUsage

Type: [DXGI_USAGE](#)

A member of the [DXGI_USAGE](#) enumerated type that describes the surface usage and CPU access options for the back buffer. The back buffer can be used for shader input or render-target output.

BufferCount

Type: [UINT](#)

A value that describes the number of buffers in the swap chain. When you call [IDXGIFactory::CreateSwapChain](#) to create a full-screen swap chain, you typically include the front buffer in this value. For more information about swap-chain buffers, see Remarks.

OutputWindow

Type: [HWND](#)

An [HWND](#) handle to the output window. This member must not be **NULL**.

Windowed

Type: [BOOL](#)

A Boolean value that specifies whether the output is in windowed mode. **TRUE** if the output is in windowed mode; otherwise, **FALSE**.

We recommend that you create a windowed swap chain and allow the end user to change the swap chain to full screen through [IDXGISwapChain::SetFullscreenState](#); that is, do not set this member to **FALSE** to force the swap chain to be full screen. However, if you create the swap chain as full screen, also provide the end user with a list of supported display modes through the **BufferDesc** member because a swap chain that is created with an unsupported display mode might cause the display to go black and prevent the end user from seeing anything.

For more information about choosing windowed versus full screen, see [IDXGIFactory::CreateSwapChain](#).

SwapEffect

Type: [DXGI_SWAP_EFFECT](#)

A member of the [DXGI_SWAP_EFFECT](#) enumerated type that describes options for handling the contents of the presentation buffer after presenting a surface.

Flags

Type: [UINT](#)

A member of the [DXGI_SWAP_CHAIN_FLAG](#) enumerated type that describes options for swap-chain behavior.

Remarks

This structure is used by the [GetDesc](#) and [CreateSwapChain](#) methods.

In full-screen mode, there is a dedicated front buffer; in windowed mode, the desktop is the front buffer.

If you create a swap chain with one buffer, specifying **DXGI_SWAP_EFFECT_SEQUENTIAL** does not cause the contents of the single buffer to be swapped with the front buffer.

For performance information about flipping swap-chain buffers in full-screen application, see [Full-Screen Application Performance Hints](#).

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Structures](#)

[IDXGIFactory::CreateSwapChain](#)

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SWAP_CHAIN_FLAG enumeration (dxgi.h)

Article01/31/2022

Options for swap-chain behavior.

Syntax

C++

```
typedef enum DXGI_SWAP_CHAIN_FLAG {
    DXGI_SWAP_CHAIN_FLAG_NONPREROTATED = 1,
    DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH = 2,
    DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE = 4,
    DXGI_SWAP_CHAIN_FLAG_RESTRICTED_CONTENT = 8,
    DXGI_SWAP_CHAIN_FLAG_RESTRICT_SHARED_RESOURCE_DRIVER = 16,
    DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY = 32,
    DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT = 64,
    DXGI_SWAP_CHAIN_FLAG_FOREGROUND_LAYER = 128,
    DXGI_SWAP_CHAIN_FLAG_FULLSCREEN_VIDEO = 256,
    DXGI_SWAP_CHAIN_FLAG_YUV_VIDEO = 512,
    DXGI_SWAP_CHAIN_FLAG_HW_PROTECTED = 1024,
    DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING = 2048,
    DXGI_SWAP_CHAIN_FLAG_RESTRICTED_TO_ALL_HOLOGRAPHIC_DISPLAYS = 4096
} ;
```

Constants

`DXGI_SWAP_CHAIN_FLAG_NONPREROTATED`

Value: 1

Set this flag to turn off automatic image rotation; that is, do not perform a rotation when transferring the contents of the front buffer to the monitor.

Use this flag to avoid a bandwidth penalty when an application expects to handle rotation. This option is valid only during full-screen mode.

`DXGI_SWAP_CHAIN_FLAG_ALLOW_MODE_SWITCH`

Value: 2

Set this flag to enable an application to switch modes by calling [IDXGISwapChain::ResizeTarget](#).

When switching from windowed to full-screen mode, the display mode (or monitor resolution) will be changed to match the dimensions of the application window.

DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE

Value: 4

Set this flag to enable an application to render using GDI on a swap chain or a surface.

This will allow the application to call [IDXGISurface1::GetDC](#) on the 0th back buffer or a surface.

This flag is not applicable for Direct3D 12.

DXGI_SWAP_CHAIN_FLAG_RESTRICTED_CONTENT

Value: 8

Set this flag to indicate that the swap chain might contain protected content; therefore, the operating system supports the creation of the swap chain only when driver and hardware protection is used. If the driver and hardware do not support content protection, the call to create a resource for the swap chain fails.

Direct3D 11: This enumeration value is supported starting with Windows 8.

DXGI_SWAP_CHAIN_FLAG_RESTRICT_SHARED_RESOURCE_DRIVER

Value: 16

Set this flag to indicate that shared resources that are created within the swap chain must be protected by using the driver's mechanism for restricting access to shared surfaces.

Direct3D 11: This enumeration value is supported starting with Windows 8.

DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY

Value: 32

Set this flag to restrict presented content to the local displays. Therefore, the presented content is not accessible via remote accessing or through the [desktop duplication APIs](#).

This flag supports the window content protection features of Windows. Applications can use this flag to protect their own onscreen window content from being captured or copied through a specific set of public operating system features and APIs.

If you use this flag with windowed ([HWND](#) or [IWindow](#)) swap chains where another process created the [HWND](#), the owner of the [HWND](#) must use the [SetWindowDisplayAffinity](#) function appropriately in order to allow calls to [IDXGISwapChain::Present](#) or [IDXGISwapChain1::Present1](#) to succeed.

Direct3D 11: This enumeration value is supported starting with Windows 8.

DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT

Value: 64

Set this flag to create a waitable object you can use to ensure rendering does not begin while a frame is still being presented. When this flag is used, the swapchain's latency must be set with the [IDXGISwapChain2::SetMaximumFrameLatency](#) API instead of [IDXGIDevice1::SetMaximumFrameLatency](#).

This flag isn't supported in full-screen mode, unless the render API is Direct3D 12.

Note This enumeration value is supported starting with Windows 8.1.

DXGI_SWAP_CHAIN_FLAG_FOREGROUND_LAYER

Value: 128

Set this flag to create a swap chain in the foreground layer for multi-plane rendering. This flag can only be used with [CoreWindow](#) swap chains, which are created with [CreateSwapChainForCoreWindow](#). Apps should not create foreground swap chains if [IDXGIOutput2::SupportsOverlays](#) indicates that hardware support for overlays is not available.

Note that [IDXGISwapChain::ResizeBuffers](#) cannot be used to add or remove this flag.

Note This enumeration value is supported starting with Windows 8.1.

DXGI_SWAP_CHAIN_FLAG_FULLSCREEN_VIDEO

Value: 256

Set this flag to create a swap chain for full-screen video.

Note This enumeration value is supported starting with Windows 8.1.

DXGI_SWAP_CHAIN_FLAG_YUV_VIDEO

Value: 512

Set this flag to create a swap chain for YUV video.

Note This enumeration value is supported starting with Windows 8.1.

`DXGI_SWAP_CHAIN_FLAG_HW_PROTECTED`

Value: 1024

Indicates that the swap chain should be created such that all underlying resources can be protected by the hardware. Resource creation will fail if hardware content protection is not supported.

This flag has the following restrictions:

- This flag can only be used with swap effect `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL`.

Note Creating a swap chain using this flag does not automatically guarantee that hardware protection will be enabled for the underlying allocation. Some implementations require that the DRM components are first initialized prior to any guarantees of protection.

Note This enumeration value is supported starting with Windows 10.

`DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING`

Value: 2048

Tearing support is a requirement to enable displays that support variable refresh rates to function properly when the application presents a swap chain tied to a full screen borderless window. Win32 apps can already achieve tearing in fullscreen exclusive mode by calling [SetFullscreenState\(TRUE\)](#), but the recommended approach for Win32 developers is to use this tearing flag instead. This flag requires the use of a `DXGI_SWAP_EFFECT_FLIP_*` swap effect.

To check for hardware support of this feature, refer to [IDXGIFactory5::CheckFeatureSupport](#). For usage information refer to [IDXGISwapChain::Present](#) and the `DXGI_PRESENT` flags.

① Note

`IDXGISwapChain::ResizeBuffers` can't be used to add or remove this flag.

`DXGI_SWAP_CHAIN_FLAG_RESTRICTED_TO_ALL_HOLOGRAPHIC_DISPLA`

Value: 4096

Remarks

This enumeration is used by the [DXGI_SWAP_CHAIN_DESC](#) structure and the [IDXGISwapChain::ResizeTarget](#) method.

This enumeration is also used by the [DXGI_SWAP_CHAIN_DESC1](#) structure.

You don't need to set [DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY](#) for swap chains that you create in full-screen mode with the [IDXGIFactory::CreateSwapChain](#) method because those swap chains already behave as if [DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY](#) is set. That is, presented content is not accessible by remote access or through the [desktop duplication APIs](#).

Swap chains that you create with the [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), and [IDXGIFactory2::CreateSwapChainForComposition](#) methods are not protected if [DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY](#) is not set and are protected if [DXGI_SWAP_CHAIN_FLAG_DISPLAY_ONLY](#) is set. When swap chains are protected, screen scraping is prevented and, in full-screen mode, presented content is not accessible through the [desktop duplication APIs](#).

When you call [IDXGISwapChain::ResizeBuffers](#) to change the swap chain's back buffer, you can reset or change all [DXGI_SWAP_CHAIN_FLAG](#) flags.

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_SWAP_EFFECT enumeration (dxgi.h)

Article 02/02/2023

Options for handling pixels in a display surface after calling [IDXGISwapChain1::Present1](#).

Syntax

C++

```
typedef enum DXGI_SWAP_EFFECT {
    DXGI_SWAP_EFFECT_DISCARD = 0,
    DXGI_SWAP_EFFECT_SEQUENTIAL = 1,
    DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL = 3,
    DXGI_SWAP_EFFECT_FLIP_DISCARD = 4
} ;
```

Constants

DXGI_SWAP_EFFECT_DISCARD

Value: 0

Use this flag to specify the bit-block transfer (bitblt) model and to specify that DXGI discard the contents of the back buffer after you call [IDXGISwapChain1::Present1](#).

This flag is valid for a swap chain with more than one back buffer, although, applications only have read and write access to buffer 0.

Use this flag to enable the display driver to select the most efficient presentation technique for the swap chain.

Direct3D 12: This enumeration value is never supported. D3D12 apps must use **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** or **DXGI_SWAP_EFFECT_FLIP_DISCARD**.

Note There are differences between full screen exclusive and full screen UWP. If you are porting a Direct3D 11 application to UWP on a Windows PC, be aware that the use of **DXGI_SWAP_EFFECT_DISCARD** when creating swap chains does not behave the same way in UWP as it does in Win32, and its use may be detrimental to GPU performance.

This is because UWP applications are forced into FLIP swap modes (even if other swap modes are set), because this reduces the computation time used by the memory copies originally done by the older bitblt model.

The recommended approach is to manually convert DX11 Discard swap chains to use flip models within UWP, using **DXGI_SWAP_EFFECT_FLIP_DISCARD** instead of **DXGI_SWAP_EFFECT_DISCARD** where possible.

Refer to the Example below, and see [this article](#) for more information.

DXGI_SWAP_EFFECT_SEQUENTIAL

Value: 1

Use this flag to specify the bitblt model and to specify that DXGI persist the contents of the back buffer after you call [IDXGISwapChain1::Present1](#).

Use this option to present the contents of the swap chain in order, from the first buffer (buffer 0) to the last buffer.

This flag cannot be used with multisampling.

Direct3D 12: This enumeration value is never supported. D3D12 apps must use **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** or **DXGI_SWAP_EFFECT_FLIP_DISCARD**.

Note For best performance, use **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** instead of **DXGI_SWAP_EFFECT_SEQUENTIAL**. See [this article](#) for more information.

DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL

Value: 3

Use this flag to specify the flip presentation model and to specify that DXGI persist the contents of the back buffer after you call [IDXGISwapChain1::Present1](#). This flag cannot be used with multisampling.

Direct3D 11: This enumeration value is supported starting with Windows 8.

`DXGI_SWAP_EFFECT_FLIP_DISCARD`

Value: 4

Use this flag to specify the flip presentation model and to specify that DXGI discard the contents of the back buffer after you call [IDXGISwapChain1::Present1](#).

This flag cannot be used with multisampling and partial presentation.

See [DXGI 1.4 Improvements](#).

Direct3D 11: This enumeration value is supported starting with Windows 10.

This flag is valid for a swap chain with more than one back buffer; although applications have read and write access only to buffer 0.

Note Windows Store apps must use `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL` or `DXGI_SWAP_EFFECT_FLIP_DISCARD`.

Remarks

This enumeration is used by the [DXGI_SWAP_CHAIN_DESC](#) and [DXGI_SWAP_CHAIN_DESC1](#) structures.

In D3D12, only `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL` and `DXGI_SWAP_EFFECT_FLIP_DISCARD` are supported, and the bitblt models are not. Because of this, multisampling a back buffer is not supported in D3D12, and you must manually perform multisampling in the app using [ID3D12GraphicsCommandList::ResolveSubresource](#) or [ID3D12GraphicsCommandList1::ResolveSubresourceRegion](#).

To use multisampling with `DXGI_SWAP_EFFECT_SEQUENTIAL` or `DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL`, you must perform the multisampling in a separate render target. For example, create a multisampled texture by calling [ID3D11Device::CreateTexture2D](#) with a filled [D3D11_TEXTURE2D_DESC](#) structure (`BindFlags` member set to `D3D11_BIND_RENDER_TARGET` and `SampleDesc` member with multisampling parameters). Next call [ID3D11Device::CreateRenderTargetView](#) to create a render-target view for the texture, and render your scene into the texture. Finally call [ID3D11DeviceContext::ResolveSubresource](#) to resolve the multisampled texture into your non-multisampled swap chain.

The primary difference between presentation models is how back-buffer contents get to the Desktop Window Manager (DWM) for composition. In the bitblt model, which is used with the **DXGI_SWAP_EFFECT_DISCARD** and **DXGI_SWAP_EFFECT_SEQUENTIAL** values, contents of the back buffer get copied into the redirection surface on each call to [IDXGISwapChain1::Present1](#). In the flip model, which is used with the **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** value, all back buffers are shared with the DWM. Therefore, the DWM can compose straight from those back buffers without any additional copy operations. In general, the flip model is the more efficient model. The flip model also provides more features, such as enhanced present statistics.

The difference between **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** and **DXGI_SWAP_EFFECT_FLIP_DISCARD** is that **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** forces DXGI to guarantee that the contents of each back buffer is preserved across [IDXGISwapChain::Present](#) calls, whereas **DXGI_SWAP_EFFECT_FLIP_DISCARD** doesn't provide this guarantee. The compositor, under certain scenarios, can use DirectFlip, where it uses the application's back buffer as the entire display back buffer, which elides the cost of copying the application's back buffer into the final desktop back buffer. With **DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL** and **DXGI_SWAP_EFFECT_FLIP_DISCARD**, this optimization can occur when the application is the only item visible on the screen. However, even when the application is not the only visible item on the screen, if the flip model is **DXGI_SWAP_EFFECT_FLIP_DISCARD**, the compositor can in some scenarios still perform this optimization, by drawing other content onto the application's back buffer.

When you call [IDXGISwapChain1::Present1](#) on a flip model swap chain (**DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL**) with 0 specified in the *SyncInterval* parameter, [IDXGISwapChain1::Present1](#)'s behavior is the same as the behavior of [Direct3D 9Ex](#)'s [IDirect3DDevice9Ex::PresentEx](#) with **D3DSWAPEFFECT_FLIPEX** and **D3DPRESENT_FORCEIMMEDIATE**. That is, the runtime not only presents the next frame instead of any previously queued frames, it also terminates any remaining time left on the previously queued frames.

Regardless of whether the flip model is more efficient, an application still might choose the bitblt model because the bitblt model is the only way to mix GDI and DirectX presentation. In the flip model, the application must create the swap chain with **DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE**, and then must use [GetDC](#) on the back buffer explicitly. After the first successful call to [IDXGISwapChain1::Present1](#) on a flip-model swap chain, GDI no longer works with the [HWND](#) that is associated with that swap chain, even after the destruction of the swap chain. This restriction even extends to methods like [ScrollWindowEx](#).

For more info about the flip-model swap chain and optimizing presentation, see [Enhancing presentation with the flip model, dirty rectangles, and scrolled areas](#).

Examples

To create a swap chain in UWP, you just need to create a new instance of the DX11 template and look at the implementation of

`DeviceResources::CreateWindowSizeDependentResources` in the [D3D12 samples](#).

syntax

```
DXGI_SWAP_CHAIN_DESC1 swapChainDesc = {0};

    swapChainDesc.Width = lround(m_d3dRenderTargetSize.Width);      // Match the size of the window.
    swapChainDesc.Height = lround(m_d3dRenderTargetSize.Height);
    swapChainDesc.Format = DXGI_FORMAT_B8G8R8A8_UNORM;           // This is the most common swap chain format.
    swapChainDesc.Stereo = false;
    swapChainDesc.SampleDesc.Count = 1;                            // Don't use multi-sampling.
    swapChainDesc.SampleDesc.Quality = 0;
    swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
    swapChainDesc.BufferCount = 2;                                // Use double-buffering to minimize latency.
    swapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_DISCARD;   // All Windows Store apps must use a flip effect.
    swapChainDesc.Flags = 2048;
    swapChainDesc.Scaling = scaling;
    swapChainDesc.AlphaMode = DXGI_ALPHA_MODE_IGNORE;

    // This sequence obtains the DXGI factory that was used to create the Direct3D device above.
    ComPtr<IDXGIDevice3> dxgiDevice;
    DX::ThrowIfFailed(m_d3dDevice.As(&dxgiDevice));

    ComPtr<IDXGIAdapter> dxgiAdapter;
    DX::ThrowIfFailed(dxgiDevice->GetAdapter(&dxgiAdapter));

    ComPtr<IDXGIFactory4> dxgiFactory;
    DX::ThrowIfFailed(dxgiAdapter-
&gt;GetParent(IID_PPV_ARGS(&dxgiFactory)));

    ComPtr<IDXGISwapChain1> swapChain;
    DX::ThrowIfFailed(
        dxgiFactory->CreateSwapChainForCoreWindow(
            m_d3dDevice.Get(),
            reinterpret_cast<IUnknown*>(&m_window.Get()),
            &swapChainDesc,
            nullptr,
            &swapChain
        )
    );
}
```

Requirements

Header	dxgi.h
--------	--------

See also

[DXGI Enumerations](#)

[For best performance, use DXGI flip model ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter interface (dxgi.h)

Article 07/22/2021

The **IDXGIAdapter** interface represents a display subsystem (including one or more GPUs, DACs and video memory).

Inheritance

The **IDXGIAdapter** interface inherits from [IDXGIOBJECT](#). **IDXGIAdapter** also has these types of members:

Methods

The **IDXGIAdapter** interface has these methods.

IDXGIAdapter::CheckInterfaceSupport
Checks whether the system supports a device interface for a graphics component.
IDXGIAdapter::EnumOutputs
Enumerate adapter (video card) outputs.
IDXGIAdapter::GetDesc
Gets a DXGI 1.0 description of an adapter (or video card).

Remarks

A display subsystem is often referred to as a video card, however, on some machines the display subsystem is part of the motherboard.

To enumerate the display subsystems, use [IDXGIFactory::EnumAdapters](#).

To get an interface to the adapter for a particular device, use [IDXGIDevice::GetAdapter](#).

To create a software adapter, use [IDXGIFactory::CreateSoftwareAdapter](#).

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter::CheckInterfaceSupport method (dxgi.h)

Article 10/13/2021

Checks whether the system supports a device interface for a graphics component.

Syntax

C++

```
HRESULT CheckInterfaceSupport(
    [in] REFGUID InterfaceName,
    [out] LARGE_INTEGER *pUMDVersion
);
```

Parameters

[in] InterfaceName

Type: [REFGUID](#)

The GUID of the interface of the device version for which support is being checked. This should usually be `_uuidof(IDXGIDevice)`, which returns the version number of the Direct3D 9 UMD (user mode driver) binary. Since WDDM 2.3, all driver components within a driver package (D3D9, D3D11, and D3D12) have been required to share a single version number, so this is a good way to query the driver version regardless of which API is being used.

[out] pUMDVersion

Type: [LARGE_INTEGER*](#)

The user mode driver version of *InterfaceName*. This is returned only if the interface is supported, otherwise this parameter will be **NULL**.

Return value

Type: [HRESULT](#)

`S_OK` indicates that the interface is supported, otherwise `DXGI_ERROR_UNSUPPORTED` is returned (For more information, see [DXGI_ERROR](#)).

Remarks

Note You can use `CheckInterfaceSupport` only to check whether a Direct3D 10.x interface is supported, and only on Windows Vista SP1 and later versions of the operating system. If you try to use `CheckInterfaceSupport` to check whether a Direct3D 11.x and later version interface is supported, `CheckInterfaceSupport` returns `DXGI_ERROR_UNSUPPORTED`. Therefore, do not use `CheckInterfaceSupport`. Instead, to verify whether the operating system supports a particular interface, try to create the interface. For example, if you call the `ID3D11Device::CreateBlendState` method and it fails, the operating system does not support the `ID3D11BlendState` interface.

Requirements

Target Platform	Windows
Header	<code>dxgi.h</code>
Library	<code>DXGI.lib</code>

See also

[DXGI Interfaces](#)

[IDXGIAAdapter](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter::EnumOutputs method (dxgi.h)

Article 10/13/2021

Enumerate adapter (video card) outputs.

Syntax

C++

```
HRESULT EnumOutputs(  
    UINT          Output,  
    [out] IDXGIOOutput **ppOutput  
>;
```

Parameters

Output

Type: [UINT](#)

The index of the output.

[out] ppOutput

Type: [IDXGIOOutput**](#)

The address of a pointer to an [IDXGIOOutput](#) interface at the position specified by the *Output* parameter.

Return value

Type: [HRESULT](#)

A code that indicates success or failure (see [DXGI_ERROR](#)). DXGI_ERROR_NOT_FOUND is returned if the index is greater than the number of outputs.

If the adapter came from a device created using D3D_DRIVER_TYPE_WARP, then the adapter has no outputs, so DXGI_ERROR_NOT_FOUND is returned.

Remarks

Note If you call this API in a Session 0 process, it returns **DXGI_ERROR_NOT_CURRENTLY_AVAILABLE**.

When the **EnumOutputs** method succeeds and fills the *ppOutput* parameter with the address of the pointer to the output interface, **EnumOutputs** increments the output interface's reference count. To avoid a memory leak, when you finish using the output interface, call the [Release](#) method to decrement the reference count.

EnumOutputs first returns the output on which the desktop primary is displayed. This output corresponds with an index of zero. **EnumOutputs** then returns other outputs.

Examples

Enumerating Outputs

Here is an example of how to use **EnumOutputs** to enumerate all the outputs on an adapter:

```
UINT i = 0;
IDXGIOoutput * pOutput;
std::vector<IDXGIOoutput*> vOutputs;
while(pAdapter->EnumOutputs(i, &pOutput) != DXGI_ERROR_NOT_FOUND)
{
    vOutputs.push_back(pOutput);
    ++i;
}
```

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIAAdapter](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter::GetDesc method (dxgi.h)

Article 10/13/2021

Gets a DXGI 1.0 description of an adapter (or video card).

Syntax

C++

```
HRESULT GetDesc(  
    [out] DXGI_ADAPTER_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [DXGI_ADAPTER_DESC*](#)

A pointer to a [DXGI_ADAPTER_DESC](#) structure that describes the adapter. This parameter must not be **NULL**. On [feature level](#) 9 graphics hardware, **GetDesc** returns zeros for the PCI ID in the **VendorId**, **DeviceId**, **SubSysId**, and **Revision** members of [DXGI_ADAPTER_DESC](#) and “Software Adapter” for the description string in the **Description** member.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise returns **E_INVALIDARG** if the *pDesc* parameter is **NULL**.

Remarks

Graphics apps can use the DXGI API to retrieve an accurate set of graphics memory values on systems that have Windows Display Driver Model (WDDM) drivers. The following are the critical steps involved.

- Graphics driver model determination —Because DXGI is only available on systems with WDDM drivers, the app must first confirm the driver model by using the

following API.

```
HasWDDMDriver()
{
    LPDIRECT3DCREATE9EX pD3D9Create9Ex = NULL;
    HMODULE             hD3D9           = NULL;

    hD3D9 = LoadLibrary( L"d3d9.dll" );

    if ( NULL == hD3D9 ) {
        return false;
    }

    /*
     * Try to create IDirect3D9Ex interface (also known as a DX9L
     * interface). This interface can only be created if the driver is a WDDM
     * driver.
     */
    //
    pD3D9Create9Ex = (LPDIRECT3DCREATE9EX) GetProcAddress( hD3D9,
    "Direct3DCreate9Ex" );

    return pD3D9Create9Ex != NULL;
}
```

- Retrieval of graphics memory values.—After the app determines the driver model to be WDDM, the app can use the Direct3D 10 or later API and DXGI to get the amount of graphics memory. After you create a Direct3D device, use this code to obtain a [DXGI_ADAPTER_DESC](#) structure that contains the amount of available graphics memory.

```
IDXGIDevice * pDXGIDevice;
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice), (void
**) &pDXGIDevice);
IDXGIAdapter * pDXGIAdapter;
pDXGIDevice->GetAdapter(&pDXGIAdapter);
DXGI_ADAPTER_DESC adapterDesc;
pDXGIAdapter->GetDesc(&adapterDesc);
```

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIAAdapter](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIAdapter1 interface (dxgi.h)

Article 08/23/2022

The **IDXGIAdapter1** interface represents a display sub-system (including one or more GPU's, DACs and video memory).

Inheritance

The **IDXGIAdapter1** interface inherits from [IDXGIAdapter](#). **IDXGIAdapter1** also has these types of members:

Methods

The **IDXGIAdapter1** interface has these methods.

[IDXGIAdapter1::GetDesc1](#)

Gets a DXGI 1.1 description of an adapter (or video card).

Remarks

This interface is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

A display sub-system is often referred to as a video card, however, on some machines the display sub-system is part of the mother board.

To enumerate the display sub-systems, use [IDXGIFactory1::EnumAdapters1](#). To get an interface to the adapter for a particular device, use [IDXGIDevice::GetAdapter](#). To create a software adapter, use [IDXGIFactory::CreateSoftwareAdapter](#).

Windows Phone 8: This API is supported.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIAAdapter](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter1::GetDesc1 method (dxgi.h)

Article 08/23/2022

Gets a DXGI 1.1 description of an adapter (or video card).

Syntax

C++

```
HRESULT GetDesc1(
    [out] DXGI_ADAPTER_DESC1 *pDesc
);
```

Parameters

[out] pDesc

Type: [DXGI_ADAPTER_DESC1*](#)

A pointer to a [DXGI_ADAPTER_DESC1](#) structure that describes the adapter. This parameter must not be **NULL**. On [feature level 9](#) graphics hardware, **GetDesc1** returns zeros for the PCI ID in the **VendorId**, **DeviceId**, **SubSysId**, and **Revision** members of [DXGI_ADAPTER_DESC1](#) and "Software Adapter" for the description string in the **Description** member.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise, returns **E_INVALIDARG** if the *pDesc* parameter is **NULL**.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Use the [GetDesc1](#) method to get a DXGI 1.1 description of an adapter. To get a DXGI 1.0 description, use the [IDXGIAdapter](#) method.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIAdapter1](#)

[IDXGIAdapter::GetDesc](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice interface (dxgi.h)

Article 07/22/2021

An **IDXGIDevice** interface implements a derived class for DXGI objects that produce image data.

Inheritance

The **IDXGIDevice** interface inherits from [IDXGIOBJECT](#). **IDXGIDevice** also has these types of members:

Methods

The **IDXGIDevice** interface has these methods.

IDXGIDevice::CreateSurface
Returns a surface. This method is used internally and you should not call it directly in your application.
IDXGIDevice::GetAdapter
Returns the adapter for the specified device.
IDXGIDevice::GetGPUThreadPriority
Gets the GPU thread priority.
IDXGIDevice::QueryResourceResidency
Gets the residency status of an array of resources.
IDXGIDevice::SetGPUThreadPriority
Sets the GPU thread priority.

Remarks

The **IDXGIDevice** interface is designed for use by DXGI objects that need access to other DXGI objects. This interface is useful to applications that do not use Direct3D to communicate with DXGI.

The Direct3D create device functions return a Direct3D device object. This Direct3D device object implements the [IUnknown](#) interface. You can query this Direct3D device object for the device's corresponding [IDXGIDevice](#) interface. To retrieve the [IDXGIDevice](#) interface of a Direct3D device, use the following code:

```
IDXGIDevice * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice), (void  
**)&pDXGIDevice);
```

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice::CreateSurface method (dxgi.h)

Article 10/13/2021

Returns a surface. This method is used internally and you should not call it directly in your application.

Syntax

C++

```
HRESULT CreateSurface(
    [in]           const DXGI_SURFACE_DESC      *pDesc,
                  UINT                         NumSurfaces,
                  DXGI_USAGE                   Usage,
    [in, optional] const DXGI_SHARED_RESOURCE *pSharedResource,
    [out]          IDXGISurface                **ppSurface
);
```

Parameters

[in] pDesc

Type: [const DXGI_SURFACE_DESC*](#)

A pointer to a [DXGI_SURFACE_DESC](#) structure that describes the surface.

NumSurfaces

Type: [UINT](#)

The number of surfaces to create.

Usage

Type: [DXGI_USAGE](#)

A [DXGI_USAGE](#) flag that specifies how the surface is expected to be used.

[in, optional] pSharedResource

Type: [const DXGI_SHARED_RESOURCE*](#)

An optional pointer to a [DXGI_SHARED_RESOURCE](#) structure that contains shared resource information for opening views of such resources.

[out] ppSurface

Type: [IDXGISurface**](#)

The address of an [IDXGISurface](#) interface pointer to the first created surface.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

The **CreateSurface** method creates a buffer to exchange data between one or more devices. It is used internally, and you should not directly call it.

The runtime automatically creates an [IDXGISurface](#) interface when it creates a Direct3D resource object that represents a surface. For example, the runtime creates an [IDXGISurface](#) interface when it calls [ID3D11Device::CreateTexture2D](#) or [ID3D10Device::CreateTexture2D](#) to create a 2D texture. To retrieve the [IDXGISurface](#) interface that represents the 2D texture surface, call [ID3D11Texture2D::QueryInterface](#) or [ID3D10Texture2D::QueryInterface](#). In this call, you must pass the identifier of [IDXGISurface](#). If the 2D texture has only a single MIP-map level and does not consist of an array of textures, [QueryInterface](#) succeeds and returns a pointer to the [IDXGISurface](#) interface pointer. Otherwise, [QueryInterface](#) fails and does not return the pointer to [IDXGISurface](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[ID3D10Device::CreateTexture2D](#)

[ID3D11Device::CreateTexture2D](#)

[IDXGIDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice::GetAdapter method (dxgi.h)

Article 10/13/2021

Returns the adapter for the specified device.

Syntax

C++

```
HRESULT GetAdapter(
    [out] IDXGIAdapter **pAdapter
);
```

Parameters

[out] pAdapter

Type: [IDXGIAdapter**](#)

The address of an [IDXGIAdapter](#) interface pointer to the adapter. This parameter must not be **NULL**.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise, returns one of the [DXGI_ERROR](#) that indicates failure. If the *pAdapter* parameter is **NULL** this method returns **E_INVALIDARG**.

Remarks

If the **GetAdapter** method succeeds, the reference count on the adapter interface will be incremented. To avoid a memory leak, be sure to release the interface when you are finished using it.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDevice::GetGPUThreadPriority method (dxgi.h)

Article 10/13/2021

Gets the GPU thread priority.

Syntax

C++

```
HRESULT GetGPUThreadPriority(  
    [out] INT *pPriority  
);
```

Parameters

[out] pPriority

Type: [INT*](#)

A pointer to a variable that receives a value that indicates the current GPU thread priority. The value will be between -7 and 7, inclusive, where 0 represents normal priority.

Return value

Type: [HRESULT](#)

Return S_OK if successful; otherwise, returns E_POINTER if the *pPriority* parameter is NULL.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDevice::QueryResourceResidency method (dxgi.h)

Article 10/13/2021

Gets the residency status of an array of resources.

Syntax

C++

```
HRESULT QueryResourceResidency(
    [in] IUnknown      * const *ppResources,
    [out] DXGI_RESIDENCY *pResidencyStatus,
    UINT             NumResources
);
```

Parameters

[in] ppResources

Type: [IUnknown*](#)

An array of [IDXGIResource](#) interfaces.

[out] pResidencyStatus

Type: [DXGI_RESIDENCY*](#)

An array of [DXGI_RESIDENCY](#) flags. Each element describes the residency status for corresponding element in the *ppResources* argument array.

NumResources

Type: [UINT](#)

The number of resources in the *ppResources* argument array and *pResidencyStatus* argument array.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, returns DXGI_ERROR_DEVICE_REMOVED, E_INVALIDARG, or E_POINTER (see [Common HRESULT Values](#) and WinError.h for more information).

Remarks

The information returned by the *pResidencyStatus* argument array describes the residency status at the time that the **QueryResourceResidency** method was called.

Note The residency status will constantly change.

If you call the **QueryResourceResidency** method during a device removed state, the *pResidencyStatus* argument will return the [DXGI_RESIDENCY_RESIDENT_IN_SHARED_MEMORY](#) flag.

Note This method should not be called every frame as it incurs a non-trivial amount of overhead.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice](#)

Feedback



Was this page helpful? [!\[\]\(84ec6d12e372dae198078201f5580e78_img.jpg\) Yes](#) [!\[\]\(e338686d151de36b75c6246aae51ca0d_img.jpg\) No](#)

[Get help at Microsoft Q&A](#)

IDXGIDevice::SetGPUThreadPriority method (dxgi.h)

Article 04/02/2021

Sets the GPU thread priority.

Syntax

C++

```
HRESULT SetGPUThreadPriority(  
    INT Priority  
);
```

Parameters

Priority

Type: [INT](#)

A value that specifies the required GPU thread priority. This value must be between -7 and 7, inclusive, where 0 represents normal priority.

Return value

Type: [HRESULT](#)

Return [S_OK](#) if successful; otherwise, returns [E_INVALIDARG](#) if the *Priority* parameter is invalid.

Remarks

The values for the *Priority* parameter function as follows:

- Positive values increase the likelihood that the GPU scheduler will grant GPU execution cycles to the device when rendering.
- Negative values lessen the likelihood that the device will receive GPU execution cycles when devices compete for them.
- The device is guaranteed to receive some GPU execution cycles at all settings.

To use the **SetGPUThreadPriority** method, you should have a comprehensive understanding of GPU scheduling. You should profile your application to ensure that it behaves as intended. If used inappropriately, the **SetGPUThreadPriority** method can impede rendering speed and result in a poor user experience.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice](#)

[IDXGIDevice::GetGPUThreadPriority](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice1 interface (dxgi.h)

Article 08/23/2022

An **IDXGIDevice1** interface implements a derived class for DXGI objects that produce image data.

Inheritance

The **IDXGIDevice1** interface inherits from [IDXGIDevice](#). **IDXGIDevice1** also has these types of members:

Methods

The **IDXGIDevice1** interface has these methods.

IDXGIDevice1::GetMaximumFrameLatency
Gets the number of frames that the system is allowed to queue for rendering.
IDXGIDevice1::SetMaximumFrameLatency
Sets the number of frames that the system is allowed to queue for rendering.

Remarks

This interface is not supported by Direct3D 12 devices. Direct3D 12 applications have direct control over their swapchain management, so better latency control should be handled by the application. You can make use of Waitable objects (refer to [DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT](#)) and the [IDXGISwapChain2::SetMaximumFrameLatency](#) method if desired.

This interface is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

The **IDXGIDevice1** interface is designed for use by DXGI objects that need access to other DXGI objects. This interface is useful to applications that do not use Direct3D to communicate with DXGI.

The Direct3D create device functions return a Direct3D device object. This Direct3D device object implements the [IUnknown](#) interface. You can query this Direct3D device object for the device's corresponding [IDXGIDevice1](#) interface. To retrieve the [IDXGIDevice1](#) interface of a Direct3D device, use the following code:

```
IDXGIDevice1 * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice1), (void  
**)&pDXGIDevice);
```

Windows Phone 8: This API is supported.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice1::GetMaximumFrameLatency method (dxgi.h)

Article 08/23/2022

Gets the number of frames that the system is allowed to queue for rendering.

Syntax

C++

```
HRESULT GetMaximumFrameLatency(
    [out] UINT *pMaxLatency
);
```

Parameters

[out] pMaxLatency

Type: [UINT*](#)

This value is set to the number of frames that can be queued for render.

This value defaults to 3, but can range from 1 to 16.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, returns one of the following members of the [D3DERR](#) enumerated type:

- D3DERR_DEVICELOST
- D3DERR_DEVICEREMOVED
- D3DERR_DRIVERINTERNALERROR
- D3DERR_INVALIDCALL
- D3DERR_OUTOFVIDEOMEMORY

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Frame latency is the number of frames that are allowed to be stored in a queue before submission for rendering. Latency is often used to control how the CPU chooses between responding to user input and frames that are in the render queue. It is often beneficial for applications that have no user input (for example, video playback) to queue more than 3 frames of data.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice1](#)

[IDXGIDevice1::SetMaximumFrameLatency](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice1::SetMaximumFrameLatency method (dxgi.h)

Article 08/23/2022

Sets the number of frames that the system is allowed to queue for rendering.

Syntax

C++

```
HRESULT SetMaximumFrameLatency(  
    UINT MaxLatency  
>;
```

Parameters

MaxLatency

Type: [UINT](#)

The maximum number of back buffer frames that a driver can queue. The value defaults to 3, but can range from 1 to 16. A value of 0 will reset latency to the default. For multi-head devices, this value is specified per-head.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, DXGI_ERROR_DEVICE_REMOVED if the device was removed.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Frame latency is the number of frames that are allowed to be stored in a queue before submission for rendering. Latency is often used to control how the CPU chooses between responding to user input and frames that are in the render queue. It is often beneficial for applications that have no user input (for example, video playback) to queue more than 3 frames of data.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDevice1](#)

[IDXGIDevice1::GetMaximumFrameLatency](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDeviceSubObject interface (dxgi.h)

Article07/22/2021

Inherited from objects that are tied to the device so that they can retrieve a pointer to it.

Inheritance

The **IDXGIDeviceSubObject** interface inherits from [IDXGIOBJECT](#). **IDXGIDeviceSubObject** also has these types of members:

Methods

The **IDXGIDeviceSubObject** interface has these methods.

IDXGIDeviceSubObject::GetDevice

Retrieves the device.

Remarks

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDeviceSubObject::GetDevice method (dxgi.h)

Article 10/13/2021

Retrieves the device.

Syntax

C++

```
HRESULT GetDevice(
    [in] REFIID riid,
    [out] void    **ppDevice
);
```

Parameters

[in] `riid`

Type: `REFIID`

The reference id for the device.

[out] `ppDevice`

Type: `void**`

The address of a pointer to the device.

Return value

Type: `HRESULT`

A code that indicates success or failure (see [DXGI_ERROR](#)).

Remarks

The type of interface that is returned can be any interface published by the device. For example, it could be an `IDXGIDevice *` called `pDevice`, and therefore the `REFIID` would be obtained by calling `_uuidof(pDevice)`.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIDeviceSubObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory interface (dxgi.h)

Article 07/22/2021

An **IDXGIFactory** interface implements methods for generating DXGI objects (which handle full screen transitions).

Inheritance

The **IDXGIFactory** interface inherits from [IDXGIOBJECT](#). **IDXGIFactory** also has these types of members:

Methods

The **IDXGIFactory** interface has these methods.

IDXGIFactory::CreateSoftwareAdapter
Create an adapter interface that represents a software adapter.
IDXGIFactory::CreateSwapChain
Creates a swap chain.
IDXGIFactory::EnumAdapters
Enumerates the adapters (video cards).
IDXGIFactory::GetWindowAssociation
Get the window through which the user controls the transition to and from full screen.
IDXGIFactory::MakeWindowAssociation
Allows DXGI to monitor an application's message queue for the alt-enter key sequence (which causes the application to switch from windowed to full screen or vice versa).

Remarks

Create a factory by calling [CreateDXGIFactory](#).

Because you can create a Direct3D device without creating a swap chain, you might need to retrieve the factory that is used to create the device in order to create a swap chain. You can request the [IDXGIDevice](#) interface from the Direct3D device and then use the [IDXGIOObject::GetParent](#) method to locate the factory. The following code shows how.

```
IDXGIDevice * pDXGIDevice = nullptr;
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice), (void **)&pDXGIDevice);

IDXGIAdapter * pDXGIAdapter = nullptr;
hr = pDXGIDevice->GetAdapter( &pDXGIAdapter );

IDXGIFactory * pIDXGIFactory = nullptr;
pDXGIAdapter->GetParent(__uuidof(IDXGIFactory), (void **)&pIDXGIFactory);
```

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIOObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory::CreateSoftwareAdapter method (dxgi.h)

Article 10/13/2021

Create an adapter interface that represents a software adapter.

Syntax

C++

```
HRESULT CreateSoftwareAdapter(
    HMODULE      Module,
    [out] IDXGIAdapter **ppAdapter
);
```

Parameters

Module

Type: [HMODULE](#)

Handle to the software adapter's dll. HMODULE can be obtained with [GetModuleHandle](#) or [LoadLibrary](#).

[out] ppAdapter

Type: [IDXGIAdapter**](#)

Address of a pointer to an adapter (see [IDXGIAdapter](#)).

Return value

Type: [HRESULT](#)

A [return code](#) indicating success or failure.

Remarks

A software adapter is a DLL that implements the entirety of a device driver interface, plus emulation, if necessary, of kernel-mode graphics components for Windows. Details

on implementing a software adapter can be found in the Windows Vista Driver Development Kit. This is a very complex development task, and is not recommended for general readers.

Calling this method will increment the module's reference count by one. The reference count can be decremented by calling [FreeLibrary](#).

The typical calling scenario is to call [LoadLibrary](#), pass the handle to [CreateSoftwareAdapter](#), then immediately call [FreeLibrary](#) on the DLL and forget the DLL's [HMODULE](#). Since the software adapter calls [FreeLibrary](#) when it is destroyed, the lifetime of the DLL will now be owned by the adapter, and the application is free of any further consideration of its lifetime.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIFactory](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory::CreateSwapChain method (dxgi.h)

Article 10/13/2021

[Starting with Direct3D 11.1, we recommend not to use [CreateSwapChain](#) anymore to create a swap chain. Instead, use [CreateSwapChainForHwnd](#), [CreateSwapChainForCoreWindow](#), or [CreateSwapChainForComposition](#) depending on how you want to create the swap chain.]

Creates a swap chain.

Syntax

C++

```
HRESULT CreateSwapChain(
    [in] IUnknown* pDevice,
    [in] DXGI_SWAP_CHAIN_DESC* pDesc,
    [out] IDXGISwapChain** ppSwapChain
);
```

Parameters

`[in] pDevice`

Type: [IUnknown*](#)

For Direct3D 11, and earlier versions of Direct3D, this is a pointer to the Direct3D device for the swap chain. For Direct3D 12 this is a pointer to a direct command queue (refer to [ID3D12CommandQueue](#)) . This parameter cannot be **NULL**.

`[in] pDesc`

Type: [DXGI_SWAP_CHAIN_DESC](#)*

A pointer to a [DXGI_SWAP_CHAIN_DESC](#) structure for the swap-chain description. This parameter cannot be **NULL**.

`[out] ppSwapChain`

Type: [IDXGISwapChain](#)**

A pointer to a variable that receives a pointer to the [IDXGISwapChain](#) interface for the swap chain that [CreateSwapChain](#) creates.

Return value

Type: [HRESULT](#)

[DXGI_ERROR_INVALID_CALL](#) if *pDesc* or *ppSwapChain* is **NULL**, [DXGI_STATUS_OCCLUDED](#) if you request full-screen mode and it is unavailable, or [E_OUTOFMEMORY](#). Other error codes defined by the type of device passed in may also be returned.

Remarks

Note If you call this API in a Session 0 process, it returns [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

If you attempt to create a swap chain in full-screen mode, and full-screen mode is unavailable, the swap chain will be created in windowed mode and [DXGI_STATUS_OCCLUDED](#) will be returned.

If the buffer width or the buffer height is zero, the sizes will be inferred from the output window size in the swap-chain description.

Because the target output can't be chosen explicitly when the swap chain is created, we recommend not to create a full-screen swap chain. This can reduce presentation performance if the swap chain size and the output window size do not match. Here are two ways to ensure that the sizes match:

- Create a windowed swap chain and then set it full-screen using [IDXGISwapChain::SetFullscreenState](#).
- Save a pointer to the swap chain immediately after creation, and use it to get the output window size during a WM_SIZE event. Then resize the swap chain buffers (with [IDXGISwapChain::ResizeBuffers](#)) during the transition from windowed to full-screen.

If the swap chain is in full-screen mode, before you release it you must use [SetFullscreenState](#) to switch it to windowed mode. For more information about releasing a swap chain, see the "Destroying a Swap Chain" section of [DXGI Overview](#).

After the runtime renders the initial frame in full screen, the runtime might unexpectedly exit full screen during a call to [IDXGISwapChain::Present](#). To work around this issue, we

recommend that you execute the following code right after you call [CreateSwapChain](#) to create a full-screen swap chain ([Windowed](#) member of [DXGI_SWAP_CHAIN_DESC](#) set to [FALSE](#)).

```
// Detect if newly created full-screen swap chain isn't actually full
// screen.
IDXGIOutput* pTarget; BOOL bFullscreen;
if (SUCCEEDED(pSwapChain->GetFullscreenState(&bFullscreen, &pTarget)))
{
    pTarget->Release();
}
else
    bFullscreen = FALSE;
// If not full screen, enable full screen again.
if (!bFullscreen)
{
    ShowWindow(hWnd, SW_MINIMIZE);
    ShowWindow(hWnd, SW_RESTORE);
    pSwapChain->SetFullscreenState(TRUE, NULL);
}
```

You can specify [DXGI_SWAP_EFFECT](#) and [DXGI_SWAP_CHAIN_FLAG](#) values in the swap-chain description that *pDesc* points to. These values allow you to use features like flip-model presentation and content protection by using pre-Windows 8 APIs.

However, to use stereo presentation and to change resize behavior for the flip model, applications must use the [IDXGIFactory2::CreateSwapChainForHwnd](#) method. Otherwise, the back-buffer contents implicitly scale to fit the presentation target size; that is, you can't turn off scaling.

Notes for Windows Store apps

If a Windows Store app calls [CreateSwapChain](#) with full screen specified, [CreateSwapChain](#) fails.

Windows Store apps call the [IDXGIFactory2::CreateSwapChainForCoreWindow](#) method to create a swap chain.

For info about how to choose a format for the swap chain's back buffer, see [Converting data for the color space](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

For best performance, use DXGI flip model

[IDXGIFactory](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory::EnumAdapters method (dxgi.h)

Article 10/13/2021

Enumerates the adapters (video cards).

Syntax

C++

```
HRESULT EnumAdapters(
    UINT          Adapter,
    [out] IDXGIAdapter **ppAdapter
);
```

Parameters

Adapter

Type: [UINT](#)

The index of the adapter to enumerate.

[out] ppAdapter

Type: [IDXGIAdapter**](#)

The address of a pointer to an [IDXGIAdapter](#) interface at the position specified by the *Adapter* parameter. This parameter must not be **NULL**.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise, returns [DXGI_ERROR_NOT_FOUND](#) if the index is greater than or equal to the number of adapters in the local system, or [DXGI_ERROR_INVALID_CALL](#) if *ppAdapter* parameter is **NULL**.

Remarks

When you create a factory, the factory enumerates the set of adapters that are available in the system. Therefore, if you change the adapters in a system, you must destroy and recreate the [IDXGIFactory](#) object. The number of adapters in a system changes when you add or remove a display card, or dock or undock a laptop.

When the **EnumAdapters** method succeeds and fills the *ppAdapter* parameter with the address of the pointer to the adapter interface, **EnumAdapters** increments the adapter interface's reference count. When you finish using the adapter interface, call the [Release](#) method to decrement the reference count before you destroy the pointer.

EnumAdapters first returns the adapter with the output on which the desktop primary is displayed. This adapter corresponds with an index of zero. **EnumAdapters** next returns other adapters with outputs. **EnumAdapters** finally returns adapters without outputs.

Examples

Enumerating Adapters

The following code example demonstrates how to enumerate adapters using the **EnumAdapters** method.

```
UINT i = 0;
IDXGIAdapter * pAdapter;
std::vector <IDXGIAdapter*> vAdapters;
while(pFactory->EnumAdapters(i, &pAdapter) != DXGI_ERROR_NOT_FOUND)
{
    vAdapters.push_back(pAdapter);
    ++i;
}
```

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory::GetWindowAssociation method (dxgi.h)

Article 10/13/2021

Get the window through which the user controls the transition to and from full screen.

Syntax

C++

```
HRESULT GetWindowAssociation(
    [out] HWND *pWindowHandle
);
```

Parameters

[out] `pWindowHandle`

Type: `HWND*`

A pointer to a window handle.

Return value

Type: `HRESULT`

Returns a code that indicates success or failure. `S_OK` indicates success, `DXGI_ERROR_INVALID_CALL` indicates `pWindowHandle` was passed in as `NULL`.

Remarks

Note If you call this API in a Session 0 process, it returns `DXGI_ERROR_NOT_CURRENTLY_AVAILABLE`.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIFactory::MakeWindowAssociation method (dxgi.h)

Article 04/02/2021

Allows DXGI to monitor an application's message queue for the alt-enter key sequence (which causes the application to switch from windowed to full screen or vice versa).

Syntax

C++

```
HRESULT MakeWindowAssociation(
    HWND WindowHandle,
    UINT Flags
);
```

Parameters

WindowHandle

Type: [HWND](#)

The handle of the window that is to be monitored. This parameter can be **NULL**; but only if *Flags* is also 0.

Flags

Type: [UINT](#)

One or more of the following values.

- **DXGI_MWA_NO_WINDOW_CHANGES** - Prevent DXGI from monitoring an applications message queue; this makes DXGI unable to respond to mode changes.
- **DXGI_MWA_NO_ALT_ENTER** - Prevent DXGI from responding to an alt-enter sequence.
- **DXGI_MWA_NO_PRINT_SCREEN** - Prevent DXGI from responding to a print-screen key.

Return value

Type: [HRESULT](#)

[DXGI_ERROR_INVALID_CALL](#) if *WindowHandle* is invalid, or [E_OUTOFMEMORY](#).

Remarks

Note If you call this API in a Session 0 process, it returns [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

The combination of *WindowHandle* and *Flags* informs DXGI to stop monitoring window messages for the previously-associated window.

If the application switches to full-screen mode, DXGI will choose a full-screen resolution to be the smallest supported resolution that is larger or the same size as the current back buffer size.

Applications can make some changes to make the transition from windowed to full screen more efficient. For example, on a WM_SIZE message, the application should release any outstanding swap-chain back buffers, call [IDXGISwapChain::ResizeBuffers](#), then re-acquire the back buffers from the swap chain(s). This gives the swap chain(s) an opportunity to resize the back buffers, and/or recreate them to enable full-screen flipping operation. If the application does not perform this sequence, DXGI will still make the full-screen/windowed transition, but may be forced to use a stretch operation (since the back buffers may not be the correct size), which may be less efficient. Even if a stretch is not required, presentation may not be optimal because the back buffers might not be directly interchangeable with the front buffer. Thus, a call to [ResizeBuffers](#) on WM_SIZE is always recommended, since WM_SIZE is always sent during a fullscreen transition.

While windowed, the application can, if it chooses, restrict the size of its window's client area to sizes to which it is comfortable rendering. A fully flexible application would make no such restriction, but UI elements or other design considerations can, of course, make this flexibility untenable. If the application further chooses to restrict its window's client area to just those that match supported full-screen resolutions, the application can field WM_SIZING, then check against [IDXGIOutput::FindClosestMatchingMode](#). If a matching mode is found, allow the resize. (The IDXGIOutput can be retrieved from [IDXGISwapChain::GetContainingOutput](#). Absent subsequent changes to desktop topology, this will be the same output that will be chosen when alt-enter is fielded and fullscreen mode is begun for that swap chain.)

Applications that want to handle mode changes or Alt+Enter themselves should call **MakeWindowAssociation** with the DXGI_MWA_NO_WINDOW_CHANGES flag after swap chain creation. The *WindowHandle* argument, if non-NULL, specifies that the application message queues will not be handled by the DXGI runtime for all swap chains of a particular target [HWND](#). Calling **MakeWindowAssociation** with the DXGI_MWA_NO_WINDOW_CHANGES flag after swapchain creation ensures that DXGI will not interfere with application's handling of window mode changes or Alt+Enter.

You must call the **MakeWindowAssociation** method on the factory object associated with the target HWND swap chain(s). You can guarantee that by calling the [IDXGIOBJECT::GetParent](#) method on the swap chain(s) to locate the factory. Here's a code example of doing that.

C++/WinRT

```
void MakeWindowAssociationWithLocatedFactory(
    winrt::com_ptr<IDXGISwapChain> const& swapChain,
    HWND hWnd,
    UINT flags)
{
    winrt::com_ptr<IDXGIFactory1> factory;
    factory.capture(swapChain, &IDXGISwapChain::GetParent);
    factory->MakeWindowAssociation(hWnd, flags);
}
```

Notes for Windows Store apps

If a Windows Store app calls **MakeWindowAssociation**, it fails with [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

A Microsoft Win32 application can use **MakeWindowAssociation** to control full-screen transitions through the Alt+Enter key combination and print screen behavior for full screen. For Windows Store apps, because DXGI can't perform full-screen transitions, a Windows Store app has no way to control full-screen transitions.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI interfaces](#)

[IDXGIFactory](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory1 interface (dxgi.h)

Article 08/23/2022

The **IDXGIFactory1** interface implements methods for generating DXGI objects.

Inheritance

The **IDXGIFactory1** interface inherits from [IDXGIFactory](#). **IDXGIFactory1** also has these types of members:

Methods

The **IDXGIFactory1** interface has these methods.

IDXGIFactory1::EnumAdapters1
Enumerates both adapters (video cards) with or without outputs.
IDXGIFactory1::IsCurrent
Informs an application of the possible need to re Enumerate adapters.

Remarks

This interface is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

To create a factory, call the [CreateDXGIFactory1](#) function.

Because you can create a Direct3D device without creating a swap chain, you might need to retrieve the factory that is used to create the device in order to create a swap chain. You can request the [IDXGIDevice](#) or [IDXGIDevice1](#) interface from the Direct3D device and then use the [IDXGIOBJECT::GetParent](#) method to locate the factory. The following code shows how.

```
IDXGIDevice1 * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice1), (void ***)&pDXGIDevice);  
  
IDXGIAdapter * pDXGIAdapter;  
hr = pDXGIDevice->GetParent(__uuidof(IDXGIAdapter), (void **)&pDXGIAdapter);  
  
IDXGIFactory1 * pIDXGIFactory;  
pDXGIAdapter->GetParent(__uuidof(IDXGIFactory1), (void **)&pIDXGIFactory);
```

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIFactory](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory1::EnumAdapters1 method (dxgi.h)

Article 08/23/2022

Enumerates both adapters (video cards) with or without outputs.

Syntax

C++

```
HRESULT EnumAdapters1(
    UINT           Adapter,
    [out] IDXGIAdapter1 **ppAdapter
);
```

Parameters

Adapter

Type: [UINT](#)

The index of the adapter to enumerate.

[out] ppAdapter

Type: [IDXGIAdapter1**](#)

The address of a pointer to an [IDXGIAdapter1](#) interface at the position specified by the *Adapter* parameter.

This parameter must not be **NULL**.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise, returns [DXGI_ERROR_NOT_FOUND](#) if the index is greater than or equal to the number of adapters in the local system, or [DXGI_ERROR_INVALID_CALL](#) if *ppAdapter* parameter is **NULL**.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

When you create a factory, the factory enumerates the set of adapters that are available in the system. Therefore, if you change the adapters in a system, you must destroy and recreate the [IDXGIFactory1](#) object. The number of adapters in a system changes when you add or remove a display card, or dock or undock a laptop.

When the [EnumAdapters1](#) method succeeds and fills the *ppAdapter* parameter with the address of the pointer to the adapter interface, [EnumAdapters1](#) increments the adapter interface's reference count. When you finish using the adapter interface, call the [Release](#) method to decrement the reference count before you destroy the pointer.

[EnumAdapters1](#) first returns the adapter with the output on which the desktop primary is displayed. This adapter corresponds with an index of zero. [EnumAdapters1](#) next returns other adapters with outputs. [EnumAdapters1](#) finally returns adapters without outputs.

Examples

Enumerating Adapters

The following code example demonstrates how to enumerate adapters using the [EnumAdapters1](#) method.

```
UINT i = 0;
IDXGIAdapter1 * pAdapter;
std::vector<IDXGIAdapter1*> vAdapters;
while(pFactory->EnumAdapters1(i, &pAdapter) != DXGI_ERROR_NOT_FOUND)
{
    vAdapters.push_back(pAdapter);
    ++i;
}
```

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory1::IsCurrent method (dxgi.h)

Article 08/23/2022

Informs an application of the possible need to re-create the factory and re-enumerate adapters.

Syntax

C++

```
BOOL IsCurrent();
```

Return value

Type: [BOOL](#)

FALSE, if a new adapter is becoming available or the current adapter is going away.
TRUE, no adapter changes.

IsCurrent returns **FALSE** to inform the calling application to re-enumerate adapters.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows

Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIKeyedMutex interface (dxgi.h)

Article 07/22/2021

Represents a keyed mutex, which allows exclusive access to a shared resource that is used by multiple devices.

Inheritance

The **IDXGIKeyedMutex** interface inherits from [IDXGIDeviceSubObject](#).

IDXGIKeyedMutex also has these types of members:

Methods

The **IDXGIKeyedMutex** interface has these methods.

IDXGIKeyedMutex::AcquireSync
Using a key, acquires exclusive rendering access to a shared resource.
IDXGIKeyedMutex::ReleaseSync
Using a key, releases exclusive rendering access to a shared resource.

Remarks

The [IDXGIFactory1](#) is required to create a resource capable of supporting the **IDXGIKeyedMutex** interface.

An **IDXGIKeyedMutex** should be retrieved for each device sharing a resource. In Direct3D 10.1, such a resource that is shared between two or more devices is created with the [D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX](#) flag. In Direct3D 11, such a resource that is shared between two or more devices is created with the [D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX](#) flag.

For information about creating a keyed mutex, see the [IDXGIKeyedMutex::AcquireSync](#) method.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIDeviceSubObject](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIKeyedMutex::AcquireSync method (dxgi.h)

Article04/02/2021

Using a key, acquires exclusive rendering access to a shared resource.

Syntax

C++

```
HRESULT AcquireSync(  
    UINT64 Key,  
    DWORD dwMilliseconds  
>;
```

Parameters

Key

Type: [UINT64](#)

A value that indicates which device to give access to. This method will succeed when the device that currently owns the surface calls the [IDXGIKeyedMutex::ReleaseSync](#) method using the same value. This value can be any [UINT64](#) value.

dwMilliseconds

Type: [DWORD](#)

The time-out interval, in milliseconds. This method will return if the interval elapses, and the keyed mutex has not been released using the specified *Key*. If this value is set to zero, the **AcquireSync** method will test to see if the keyed mutex has been released and returns immediately. If this value is set to **INFINITE**, the time-out interval will never elapse.

Return value

Type: [HRESULT](#)

Return **S_OK** if successful.

If the owning device attempted to create another keyed mutex on the same shared resource, **AcquireSync** returns E_FAIL.

AcquireSync can also return the following **DWORD** constants. Therefore, you should explicitly check for these constants. If you only use the **SUCCEEDED** macro on the return value to determine if **AcquireSync** succeeded, you will not catch these constants.

- WAIT_ABANDONED - The shared surface and keyed mutex are no longer in a consistent state. If **AcquireSync** returns this value, you should release and recreate both the keyed mutex and the shared surface.
- WAIT_TIMEOUT - The time-out interval elapsed before the specified key was released.

Remarks

The **AcquireSync** method creates a lock to a surface that is shared between multiple devices, allowing only one device to render to a surface at a time.

This method uses a key to determine which device currently has exclusive access to the surface.

When a surface is created using the **D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX** value of the **D3D10_RESOURCE_MISC_FLAG** enumeration, you must call the **AcquireSync** method before rendering to the surface. You must call the **ReleaseSync** method when you are done rendering to a surface.

To acquire a reference to the keyed mutex object of a shared resource, call the **QueryInterface** method of the resource and pass in the **UUID** of the **IDXGIKeyedMutex** interface. For more information about acquiring this reference, see the following code example.

The **AcquireSync** method uses the key as follows, depending on the state of the surface:

- On initial creation, the surface is unowned and any device can call the **AcquireSync** method to gain access. For an unowned device, only a key of 0 will succeed. Calling the **AcquireSync** method for any other key will stall the calling CPU thread.
- If the surface is owned by a device when you call the **AcquireSync** method, the CPU thread that called the **AcquireSync** method will stall until the owning device calls the **ReleaseSync** method using the same Key.
- If the surface is unowned when you call the **AcquireSync** method (for example, the last owning device has already called the **ReleaseSync** method), the **AcquireSync** method will succeed if you specify the same key that was specified when the

`ReleaseSync` method was last called. Calling the `AcquireSync` method using any other key will cause a stall.

- When the owning device calls the `ReleaseSync` method with a particular key, and more than one device is waiting after calling the `AcquireSync` method using the same key, any one of the waiting devices could be woken up first. The order in which devices are woken up is undefined.
- A keyed mutex does not support recursive calls to the `AcquireSync` method.

Examples

Acquiring a Keyed Mutex

The following code example demonstrates how to acquire a lock to a shared resource and how to specify a key upon release.

```
// pDesc has already been set up with texture description.  
pDesc.MiscFlags = D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX;  
  
// Create a shared texture resource.  
pD3D10DeviceD->CreateTexture2D(pDesc, NULL, pD3D10Texture);  
  
// Acquire a reference to the keyed mutex.  
pD3D10Texture->QueryInterface(_uuidof(IDXGIKeyedMutex), pDXGIKeyedMutex);  
  
// Acquire a lock to the resource.  
pDXGIKeyedMutex->AcquireSync(0, INFINITE);  
  
// Release the lock and specify a key.  
pDXGIKeyedMutex->ReleaseSync(1);
```

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory2](#)

[IDXGIFactory2::GetDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIKeyedMutex::ReleaseSync method (dxgi.h)

Article04/02/2021

Using a key, releases exclusive rendering access to a shared resource.

Syntax

C++

```
HRESULT ReleaseSync(  
    UINT64 Key  
);
```

Parameters

Key

Type: [UINT64](#)

A value that indicates which device to give access to. This method succeeds when the device that currently owns the surface calls the **ReleaseSync** method using the same value. This value can be any [UINT64](#) value.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful.

If the device attempted to release a keyed mutex that is not valid or owned by the device, **ReleaseSync** returns [E_FAIL](#).

Remarks

The **ReleaseSync** method releases a lock to a surface that is shared between multiple devices. This method uses a key to determine which device currently has exclusive access to the surface.

When a surface is created using the `D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX` value of the `D3D10_RESOURCE_MISC_FLAG` enumeration, you must call the `IDXGIKeyedMutex::AcquireSync` method before rendering to the surface. You must call the `ReleaseSync` method when you are done rendering to a surface.

After you call the `ReleaseSync` method, the shared resource is unset from the rendering pipeline.

To acquire a reference to the keyed mutex object of a shared resource, call the `QueryInterface` method of the resource and pass in the `UUID` of the `IDXGIKeyedMutex` interface. For more information about acquiring this reference, see the following code example.

Examples

Acquiring a Keyed Mutex

The following code example demonstrates how to acquire a lock to a shared resource and how to specify a key upon release.

```
// pDesc has already been set up with texture description.  
pDesc.MiscFlags = D3D10_RESOURCE_MISC_SHARED_KEYEDMUTEX;  
  
// Create a shared texture resource.  
pD3D10DeviceD->CreateTexture2D(pDesc, NULL, pD3D10Texture);  
  
// Acquire a reference to the keyed mutex.  
pD3D10Texture->QueryInterface(_uuidof(IDXGIKeyedMutex), pDXGIKeyedMutex);  
  
// Acquire a lock to the resource.  
pDXGIKeyedMutex->AcquireSync(0, INFINITE);  
  
// Release the lock and specify a key.  
pDXGIKeyedMutex->ReleaseSync(1);
```

Requirements

Target Platform	Windows
Header	dxgi.h

Library

DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory](#)

[IDXGIFactory::GetDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOBJECT interface (dxgi.h)

Article 07/22/2021

An **IDXGIOBJECT** interface is a base interface for all DXGI objects; **IDXGIOBJECT** supports associating caller-defined (private data) with an object and retrieval of an interface to the parent object.

Inheritance

The **IDXGIOBJECT** interface inherits from the [IUnknown](#) interface. **IDXGIOBJECT** also has these types of members:

Methods

The **IDXGIOBJECT** interface has these methods.

IDXGIOBJECT::GetParent
Gets the parent of the object.
IDXGIOBJECT::GetPrivateData
Get a pointer to the object's data.
IDXGIOBJECT::SetPrivateData
Sets application-defined data to the object and associates that data with a GUID.
IDXGIOBJECT::SetPrivateDataInterface
Set an interface in the object's private data.

Remarks

IDXGIOBJECT implements base-class functionality for the following interfaces:

- [IDXGIAdapter](#)
- [IDXGIDevice](#)
- [IDXGIFactory](#)
- [IDXGIOOutput](#)

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOBJECT::GetParent method (dxgi.h)

Article 10/13/2021

Gets the parent of the object.

Syntax

C++

```
HRESULT GetParent(  
    [in] REFIID riid,  
    [out] void    **ppParent  
) ;
```

Parameters

[in] riid

Type: **REFIID**

The ID of the requested interface.

[out] ppParent

Type: **void****

The address of a pointer to the parent object.

Return value

Type: **HRESULT**

Returns one of the **DXGI_ERROR** values.

Requirements

Target Platform	Windows
Header	dxgi.h

Library

DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOBJECT::GetPrivateData method (dxgi.h)

Article 10/13/2021

Get a pointer to the object's data.

Syntax

C++

```
HRESULT GetPrivateData(  
    [in]      REFGUID Name,  
    [in, out]  UINT     *pDataSize,  
    [out]     void     *pData  
>;
```

Parameters

[in] Name

Type: [REFGUID](#)

A GUID identifying the data.

[in, out] pDataSize

Type: [UINT*](#)

The size of the data.

[out] pData

Type: [void*](#)

Pointer to the data.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

If the data returned is a pointer to an [IUnknown](#), or one of its derivative classes, previously set by [IDXGIOBJECT::SetPrivateDataInterface](#), you must call [::Release\(\)](#) on the pointer before the pointer is freed to decrement the reference count.

You can pass **GUID_DeviceType** in the *Name* parameter of [GetPrivateData](#) to retrieve the device type from the display adapter object ([IDXGIAdapter](#), [IDXGIAdapter1](#), [IDXGIAdapter2](#)).

To get the type of device on which the display adapter was created

1. Call [IUnknown::QueryInterface](#) on the [ID3D11Device](#) or [ID3D10Device](#) object to retrieve the [IDXGIDevice](#) object.
2. Call [GetParent](#) on the [IDXGIDevice](#) object to retrieve the [IDXGIAdapter](#) object.
3. Call [GetPrivateData](#) on the [IDXGIAdapter](#) object with **GUID_DeviceType** to retrieve the type of device on which the display adapter was created. *pData* will point to a value from the driver-type enumeration (for example, a value from [D3D_DRIVER_TYPE](#)).

On Windows 7 or earlier, this type is either a value from [D3D10_DRIVER_TYPE](#) or [D3D_DRIVER_TYPE](#) depending on which kind of device was created. On Windows 8, this type is always a value from [D3D_DRIVER_TYPE](#). Don't use [IDXGIOBJECT::SetPrivateData](#) with **GUID_DeviceType** because the behavior when doing so is undefined.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOBJECT::SETPRIVATEDATA METHOD (dxgi.h)

Article 10/13/2021

Sets application-defined data to the object and associates that data with a GUID.

Syntax

C++

```
HRESULT SetPrivateData(
    [in] REFGUID     Name,
    [in]          UINT   DataSize,
    [in] const void *pData
);
```

Parameters

[in] Name

Type: [REFGUID](#)

A GUID that identifies the data. Use this GUID in a call to [GetPrivateData](#) to get the data.

DataSize

Type: [UINT](#)

The size of the object's data.

[in] pData

Type: [const void*](#)

A pointer to the object's data.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

SetPrivateData makes a copy of the specified data and stores it with the object.

Private data that **SetPrivateData** stores in the object occupies the same storage space as private data that is stored by associated Direct3D objects (for example, by a Microsoft Direct3D 11 device through [ID3D11Device::SetPrivateData](#) or by a Direct3D 11 child device through [ID3D11DeviceChild::SetPrivateData](#)).

The [debug layer](#) reports memory leaks by outputting a list of object interface pointers along with their friendly names. The default friendly name is "<unnamed>". You can set the friendly name so that you can determine if the corresponding object interface pointer caused the leak. To set the friendly name, use the **SetPrivateData** method and the well-known private data GUID (**WKP DID_D3D Debug Object Name**) that is in D3Dcommon.h. For example, to give pContext a friendly name of *My name*, use the following code:

```
static const char c_szName[] = "My name";
hr = pContext->SetPrivateData( WKP DID_D3D Debug Object Name, sizeof( c_szName )
- 1, c_szName );
```

You can use **WKP DID_D3D Debug Object Name** to track down memory leaks and understand performance characteristics of your applications. This information is reflected in the output of the [debug layer](#) that is related to memory leaks ([ID3D11Debug::ReportLiveDeviceObjects](#)) and with the [event tracing](#) for Windows events that we've added to Windows 8.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOBJECT::SETPRIVATEDATAINTERFACE method (dxgi.h)

Article 10/13/2021

Set an interface in the object's private data.

Syntax

C++

```
HRESULT SetPrivateDataInterface(  
    [in] REFGUID      Name,  
    [in] const IUnknown *pUnknown  
>;
```

Parameters

[in] Name

Type: [REFGUID](#)

A GUID identifying the interface.

[in] pUnknown

Type: [const IUnknown*](#)

The interface to set.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

This API associates an interface pointer with the object.

When the interface is set its reference count is incremented. When the data are overwritten (by calling SPD or SPDI with the same GUID) or the object is destroyed, ::Release() is called and the interface's reference count is decremented.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIOBJECT](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput interface (dxgi.h)

Article 07/27/2022

An **IDXGIOutput** interface represents an adapter output (such as a monitor).

Inheritance

The **IDXGIOutput** interface inherits from [IDXGIOBJECT](#). **IDXGIOutput** also has these types of members:

Methods

The **IDXGIOutput** interface has these methods.

IDXGIOutput::FindClosestMatchingMode	Finds the display mode that most closely matches the requested display mode. (<code>IDXGIOutput.FindClosestMatchingMode</code>)
IDXGIOutput::GetDesc	Get a description of the output.
IDXGIOutput::GetDisplayModeList	Gets the display modes that match the requested format and other input options. (<code>IDXGIOutput.GetDisplayModeList</code>)
IDXGIOutput::GetDisplaySurfaceData	Gets a copy of the current display surface.
IDXGIOutput::GetFrameStatistics	Gets statistics about recently rendered frames.
IDXGIOutput::GetGammaControl	Gets the gamma control settings.
IDXGIOutput::GetGammaControlCapabilities	Gets a description of the gamma-control capabilities.

[IDXGIOOutput::ReleaseOwnership](#)

Releases ownership of the output.

[IDXGIOOutput::SetDisplaySurface](#)

Changes the display mode.

[IDXGIOOutput::SetGammaControl](#)

Sets the gamma controls.

[IDXGIOOutput::TakeOwnership](#)

Takes ownership of an output.

[IDXGIOOutput::WaitForVBlank](#)

Halt a thread until the next vertical blank occurs.

Remarks

To see the outputs available, use [IDXGIAdapter::EnumOutputs](#). To see the specific output that the swap chain will update, use [IDXGISwapChain::GetContainingOutput](#).

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIOObject](#)

Feedback

Was this page helpful?  Yes  No

Get help at Microsoft Q&A

IDXGIOutput::FindClosestMatchingMode method (dxgi.h)

Article 07/27/2022

[Starting with Direct3D 11.1, we recommend not to use **FindClosestMatchingMode** anymore to find the display mode that most closely matches the requested display mode. Instead, use [IDXGIOOutput1::FindClosestMatchingMode1](#), which supports stereo display mode.]

Finds the display mode that most closely matches the requested display mode.

Syntax

C++

```
HRESULT FindClosestMatchingMode(
    [in]           const DXGI_MODE_DESC *pModeToMatch,
    [out]          DXGI_MODE_DESC       *pClosestMatch,
    [in, optional] IUnknown            *pConcernedDevice
);
```

Parameters

[in] `pModeToMatch`

Type: [const DXGI_MODE_DESC*](#)

The desired display mode (see [DXGI_MODE_DESC](#)). Members of **DXGI_MODE_DESC** can be unspecified indicating no preference for that member. A value of 0 for **Width** or **Height** indicates the value is unspecified. If either **Width** or **Height** are 0, both must be 0. A numerator and denominator of 0 in **RefreshRate** indicate it is unspecified. Other members of **DXGI_MODE_DESC** have enumeration values indicating the member is unspecified. If *pConcernedDevice* is **NULL**, **Format** cannot be **DXGI_FORMAT_UNKNOWN**.

[out] `pClosestMatch`

Type: [DXGI_MODE_DESC*](#)

The mode that most closely matches *pModeToMatch*.

[in, optional] `pConcernedDevice`

Type: [IUnknown](#)*

A pointer to the Direct3D device interface. If this parameter is **NULL**, only modes whose format matches that of *pModeToMatch* will be returned; otherwise, only those formats that are supported for scan-out by the device are returned. For info about the formats that are supported for scan-out by the device at each feature level:

- [DXGI Format Support for Direct3D Feature Level 12.1 Hardware](#)
- [DXGI Format Support for Direct3D Feature Level 12.0 Hardware](#)
- [DXGI Format Support for Direct3D Feature Level 11.1 Hardware](#)
- [DXGI Format Support for Direct3D Feature Level 11.0 Hardware](#)
- [Hardware Support for Direct3D 10Level9 Formats](#)
- [Hardware Support for Direct3D 10.1 Formats](#)
- [Hardware Support for Direct3D 10 Formats](#)

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

FindClosestMatchingMode behaves similarly to the [IDXGIOutput1::FindClosestMatchingMode1](#) except **FindClosestMatchingMode** considers only the mono display modes. [IDXGIOutput1::FindClosestMatchingMode1](#) considers only stereo modes if you set the **Stereo** member in the [DXGI_MODE_DESC1](#) structure that *pModeToMatch* points to, and considers only mono modes if **Stereo** is not set.

[IDXGIOutput1::FindClosestMatchingMode1](#) returns a matched display-mode set with only stereo modes or only mono modes. **FindClosestMatchingMode** behaves as though you specified the input mode as mono.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetDesc method (dxgi.h)

Article 10/13/2021

Get a description of the output.

Syntax

C++

```
HRESULT GetDesc(
    [out] DXGI_OUTPUT_DESC *pDesc
);
```

Parameters

[out] pDesc

Type: [DXGI_OUTPUT_DESC*](#)

A pointer to the output description (see [DXGI_OUTPUT_DESC](#)).

Return value

Type: [HRESULT](#)

Returns a code that indicates success or failure. S_OK if successful, [DXGI_ERROR_INVALID_CALL](#) if *pDesc* is passed in as NULL.

Remarks

On a high DPI desktop, **GetDesc** returns the visualized screen size unless the app is marked high DPI aware. For info about writing DPI-aware Win32 apps, see [High DPI](#).

Requirements

Target Platform	Windows
Header	dxgi.h

Library

DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetDisplayModeList method (dxgi.h)

Article 07/27/2022

[Starting with Direct3D 11.1, we recommend not to use [GetDisplayModeList](#) anymore to retrieve the matching display mode. Instead, use [IDXGIOutput1::GetDisplayModeList1](#), which supports stereo display mode.]

Gets the display modes that match the requested format and other input options.

Syntax

C++

```
HRESULT GetDisplayModeList(
    DXGI_FORMAT     EnumFormat,
    UINT            Flags,
    [in, out]        UINT          *pNumModes,
    [out, optional] DXGI_MODE_DESC *pDesc
);
```

Parameters

`EnumFormat`

Type: [DXGI_FORMAT](#)

The color format (see [DXGI_FORMAT](#)).

`Flags`

Type: [UINT](#)

Options for modes to include (see [DXGI_ENUM_MODES](#)). `DXGI_ENUM_MODES_SCALING` needs to be specified to expose the display modes that require scaling. Centered modes, requiring no scaling and corresponding directly to the display output, are enumerated by default.

`[in, out] pNumModes`

Type: [UINT*](#)

Set *pDesc* to **NULL** so that *pNumModes* returns the number of display modes that match the format and the options. Otherwise, *pNumModes* returns the number of display modes returned in *pDesc*.

[out, optional] *pDesc*

Type: [DXGI_MODE_DESC*](#)

A pointer to a list of display modes (see [DXGI_MODE_DESC](#)); set to **NULL** to get the number of display modes.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#). It is rare, but possible, that the display modes available can change immediately after calling this method, in which case [DXGI_ERROR_MORE_DATA](#) is returned (if there is not enough room for all the display modes).

If [GetDisplayModeList](#) is called from a Remote Desktop Services session (formerly Terminal Services session), [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#) is returned.

Remarks

In general, when switching from windowed to full-screen mode, a swap chain automatically chooses a display mode that meets (or exceeds) the resolution, color depth and refresh rate of the swap chain. To exercise more control over the display mode, use this API to poll the set of display modes that are validated against monitor capabilities, or all modes that match the desktop (if the desktop settings are not validated against the monitor).

As shown, this API is designed to be called twice. First to get the number of modes available, and second to return a description of the modes.

```
UINT num = 0;
DXGI_FORMAT format = DXGI_FORMAT_R32G32B32A32_FLOAT;
UINT flags       = DXGI_ENUM_MODES_INTERLACED;

p0Output->GetDisplayModeList( format, flags, &num, 0);

...
```

```
DXGI_MODE_DESC * pDescs = new DXGI_MODE_DESC[num];
pOutput->GetDisplayModeList( format, flags, &num, pDescs);
```

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetDisplaySurfaceData method (dxgi.h)

Article 10/13/2021

[Starting with Direct3D 11.1, we recommend not to use [GetDisplaySurfaceData](#) anymore to retrieve the current display surface. Instead, use [IDXGIOOutput1::GetDisplaySurfaceData1](#), which supports stereo display mode.]

Gets a copy of the current display surface.

Syntax

C++

```
HRESULT GetDisplaySurfaceData(  
    [in] IDXGISurface *pDestination  
);
```

Parameters

`[in] pDestination`

Type: [IDXGISurface*](#)

A pointer to a destination surface (see [IDXGISurface](#)).

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

[IDXGIOutput::GetDisplaySurfaceData](#) can only be called when an output is in full-screen mode. If the method succeeds, DXGI fills the destination surface.

Use [IDXGIOOutput::GetDesc](#) to determine the size (width and height) of the output when you want to allocate space for the destination surface. This is true regardless of target

monitor rotation. A destination surface created by a graphics component (such as Direct3D 10) must be created with CPU-write permission (see D3D10_CPU_ACCESS_WRITE). Other surfaces should be created with CPU read-write permission (see D3D10_CPU_ACCESS_READ_WRITE). This method will modify the surface data to fit the destination surface (stretch, shrink, convert format, rotate). The stretch and shrink is performed with point-sampling.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetFrameStatistics method (dxgi.h)

Article 10/13/2021

Gets statistics about recently rendered frames.

Syntax

C++

```
HRESULT GetFrameStatistics(  
    [out] DXGI_FRAME_STATISTICS *pStats  
)
```

Parameters

[out] pStats

Type: [DXGI_FRAME_STATISTICS*](#)

A pointer to frame statistics (see [DXGI_FRAME_STATISTICS](#)).

Return value

Type: [HRESULT](#)

If this function succeeds, it returns S_OK. Otherwise, it might return [DXGI_ERROR_INVALID_CALL](#).

Remarks

This API is similar to [IDXGISwapChain::GetFrameStatistics](#).

Note Calling this method is only supported while in full-screen mode.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetGammaControl method (dxgi.h)

Article 10/13/2021

Gets the gamma control settings.

Syntax

C++

```
HRESULT GetGammaControl(
    [out] DXGI_GAMMA_CONTROL *pArray
);
```

Parameters

[out] pArray

Type: [DXGI_GAMMA_CONTROL*](#)

An array of gamma control settings (see [DXGI_GAMMA_CONTROL](#)).

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

Note Calling this method is only supported while in full-screen mode.

For info about using gamma correction, see [Using gamma correction](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::GetGammaControlCapabilities method (dxgi.h)

Article 10/13/2021

Gets a description of the gamma-control capabilities.

Syntax

C++

```
HRESULT GetGammaControlCapabilities(
    [out] DXGI_GAMMA_CONTROL_CAPABILITIES *pGammaCaps
);
```

Parameters

[out] pGammaCaps

Type: [DXGI_GAMMA_CONTROL_CAPABILITIES*](#)

A pointer to a description of the gamma-control capabilities (see [DXGI_GAMMA_CONTROL_CAPABILITIES](#)).

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

Note Calling this method is only supported while in full-screen mode.

For info about using gamma correction, see [Using gamma correction](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::ReleaseOwnership method (dxgi.h)

Article 06/29/2021

Releases ownership of the output.

Syntax

C++

```
void ReleaseOwnership();
```

Return value

None

Remarks

If you are not using a swap chain, get access to an output by calling [IDXGIOutput::TakeOwnership](#) and release it when you are finished by calling [IDXGIOutput::ReleaseOwnership](#). An application that uses a swap chain will typically not call either of these methods.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DxGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::SetDisplaySurface method (dxgi.h)

Article 10/13/2021

Changes the display mode.

Syntax

C++

```
HRESULT SetDisplaySurface(  
    [in] IDXGISurface *pScanoutSurface  
);
```

Parameters

[in] pScanoutSurface

Type: [IDXGISurface*](#)

A pointer to a surface (see [IDXGISurface](#)) used for rendering an image to the screen. The surface must have been created as a back buffer (DXGI_USAGE_BACKBUFFER).

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

`IDXGIOutput::SetDisplaySurface` should not be called directly by applications, since results will be unpredictable. It is called implicitly by the DXGI swap chain object during full-screen transitions, and should not be used as a substitute for swap-chain methods.

This method should only be called between [IDXGIOutput::TakeOwnership](#) and [IDXGIOutput::ReleaseOwnership](#) calls.

Notes for Windows Store apps

If a Windows Store app uses `SetDisplaySurface`, it fails with `DXGI_ERROR_NOT_CURRENTLY_AVAILABLE`.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIOutput::SetGammaControl method (dxgi.h)

Article 10/13/2021

Sets the gamma controls.

Syntax

C++

```
HRESULT SetGammaControl(  
    [in] const DXGI_GAMMA_CONTROL *pArray  
);
```

Parameters

[in] pArray

Type: [const DXGI_GAMMA_CONTROL*](#)

A pointer to a [DXGI_GAMMA_CONTROL](#) structure that describes the gamma curve to set.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

Note Calling this method is only supported while in full-screen mode.

For info about using gamma correction, see [Using gamma correction](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::TakeOwnership method (dxgi.h)

Article 10/13/2021

Takes ownership of an output.

Syntax

C++

```
HRESULT TakeOwnership(
    [in] IUnknown *pDevice,
    BOOL          Exclusive
);
```

Parameters

[in] pDevice

Type: [IUnknown*](#)

A pointer to the [IUnknown](#) interface of a device (such as an [ID3D10Device](#)).

Exclusive

Type: [BOOL](#)

Set to **TRUE** to enable other threads or applications to take ownership of the device; otherwise, set to **FALSE**.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

When you are finished with the output, call [IDXGIOutput::ReleaseOwnership](#).

`TakeOwnership` should not be called directly by applications, since results will be unpredictable. It is called implicitly by the DXGI swap chain object during full-screen transitions, and should not be used as a substitute for swap-chain methods.

Notes for Windows Store apps

If a Windows Store app uses `TakeOwnership`, it fails with [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput::WaitForVBlank method (dxgi.h)

Article 06/29/2021

Halt a thread until the next vertical blank occurs.

Syntax

C++

```
HRESULT WaitForVBlank();
```

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

A vertical blank occurs when the raster moves from the lower right corner to the upper left corner to begin drawing the next frame.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource interface (dxgi.h)

Article07/22/2021

An **IDXGIResource** interface allows resource sharing and identifies the memory that a resource resides in.

Inheritance

The **IDXGIResource** interface inherits from [IDXGIDeviceSubObject](#). **IDXGIResource** also has these types of members:

Methods

The **IDXGIResource** interface has these methods.

IDXGIResource::GetEvictionPriority
Get the eviction priority.
IDXGIResource::GetSharedHandle
Gets the handle to a shared resource.
IDXGIResource::GetUsage
Get the expected resource usage.
IDXGIResource::SetEvictionPriority
Set the priority for evicting the resource from memory.

Remarks

To find out what type of memory a resource is currently located in, use [IDXGIDevice::QueryResourceResidency](#). To share resources between processes, use [ID3D10Device::OpenSharedResource](#). For information about how to share resources between multiple Windows graphics APIs, including Direct3D 11, Direct2D, Direct3D 10, and Direct3D 9Ex, see [Surface Sharing Between Windows Graphics APIs](#).

You can retrieve the **IDXGIResource** interface from any video memory resource that you create from a Direct3D 10 and later function. Any Direct3D object that supports **ID3D10Resource** or **ID3D11Resource** also supports **IDXGIResource**. For example, the Direct3D 2D texture object that you create from **ID3D11Device::CreateTexture2D** supports **IDXGIResource**. You can call **QueryInterface** on the 2D texture object (**ID3D11Texture2D**) to retrieve the **IDXGIResource** interface. For example, to retrieve the **IDXGIResource** interface from the 2D texture object, use the following code.

```
IDXGIResource * pDXGIResource;  
hr = g_pd3dTexture2D->QueryInterface(__uuidof(IDXGIResource), (void  
**)&pDXGIResource);
```

Windows Phone 8: This API is supported.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIDeviceSubObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource::GetEvictionPriority method (dxgi.h)

Article 10/13/2021

Get the eviction priority.

Syntax

C++

```
HRESULT GetEvictionPriority(  
    [out] UINT *pEvictionPriority  
);
```

Parameters

[out] pEvictionPriority

Type: [UINT*](#)

A pointer to the eviction priority, which determines when a resource can be evicted from memory.

The following defined values are possible.

Value	Meaning
<code>DXGI_RESOURCE_PRIORITY_MINIMUM</code> (0x28000000)	The resource is unused and can be evicted as soon as another resource requires the memory that the resource occupies.
<code>DXGI_RESOURCE_PRIORITY_LOW</code> (0x50000000)	The eviction priority of the resource is low. The placement of the resource is not critical, and minimal work is performed to find a location for the resource. For example, if a GPU can render with a vertex buffer from either local or non-local memory with little difference in performance, that vertex buffer is low priority. Other more critical resources (for example, a render target or texture) can then occupy the faster memory.
<code>DXGI_RESOURCE_PRIORITY_NORMAL</code> (0x78000000)	The eviction priority of the resource is normal. The placement of the resource is important, but not critical.

	for performance. The resource is placed in its preferred location instead of a low-priority resource.
DXGI_RESOURCE_PRIORITY_HIGH (0xa0000000)	The eviction priority of the resource is high. The resource is placed in its preferred location instead of a low-priority or normal-priority resource.
DXGI_RESOURCE_PRIORITY_MAXIMUM (0xc8000000)	The resource is evicted from memory only if there is no other way of resolving the memory requirement.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

The eviction priority is a memory-management variable that is used by DXGI to determine how to manage overcommitted memory.

Priority levels other than the defined values are used when appropriate. For example, a resource with a priority level of 0x78000001 indicates that the resource is slightly above normal.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIResource](#)

Feedback



Was this page helpful?  

Get help at Microsoft Q&A

IDXGIResource::GetSharedHandle method (dxgi.h)

Article 10/13/2021

[Starting with Direct3D 11.1, we recommend not to use [GetSharedHandle](#) anymore to retrieve the handle to a shared resource. Instead, use [IDXGIResource1::CreateSharedHandle](#) to get a handle for sharing. To use [IDXGIResource1::CreateSharedHandle](#), you must create the resource as shared and specify that it uses NT handles (that is, you set the [D3D11_RESOURCE_MISC_SHARED_NTHANDLE](#) flag). We also recommend that you create shared resources that use NT handles so you can use [CloseHandle](#), [DuplicateHandle](#), and so on on those shared resources.]

Gets the handle to a shared resource.

Syntax

C++

```
HRESULT GetSharedHandle(  
    [out] HANDLE *pSharedHandle  
) ;
```

Parameters

`[out] pSharedHandle`

Type: [HANDLE*](#)

A pointer to a handle.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

GetSharedHandle returns a handle for the resource that you created as shared (that is, you set the `D3D11_RESOURCE_MISC_SHARED` with or without the `D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX` flag). You can pass this handle to the `ID3D11Device::OpenSharedResource` method to give another device access to the shared resource. You can also marshal this handle to another process to share a resource with a device in another process. However, this handle is not an NT handle. Therefore, don't use the handle with `CloseHandle`, `DuplicateHandle`, and so on.

The creator of a shared resource must not destroy the resource until all intended entities have opened the resource. The validity of the handle is tied to the lifetime of the underlying video memory. If no resource objects exist on any devices that refer to this resource, the handle is no longer valid. To extend the lifetime of the handle and video memory, you must open the shared resource on a device.

GetSharedHandle can also return handles for resources that were passed into `ID3D11Device::OpenSharedResource` to open those resources.

GetSharedHandle fails if the resource to which it wants to get a handle is not shared.

Requirements

Target Platform	Windows
Header	<code>dxgi.h</code>
Library	<code>DXGI.lib</code>

See also

[IDXGIResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource::GetUsage method (dxgi.h)

Article04/02/2021

Get the expected resource usage.

Syntax

C++

```
HRESULT GetUsage(  
    DXGI_USAGE *pUsage  
) ;
```

Parameters

pUsage

Type: [DXGI_USAGE*](#)

A pointer to a usage flag (see [DXGI_USAGE](#)). For Direct3D 10, a surface can be used as a shader input or a render-target output.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource::SetEvictionPriority method (dxgi.h)

Article04/02/2021

Set the priority for evicting the resource from memory.

Syntax

C++

```
HRESULT SetEvictionPriority(  
    UINT EvictionPriority  
) ;
```

Parameters

EvictionPriority

Type: [UINT](#)

The priority is one of the following values:

Value	Meaning
<code>DXGI_RESOURCE_PRIORITY_MINIMUM</code> (0x28000000)	The resource is unused and can be evicted as soon as another resource requires the memory that the resource occupies.
<code>DXGI_RESOURCE_PRIORITY_LOW</code> (0x50000000)	The eviction priority of the resource is low. The placement of the resource is not critical, and minimal work is performed to find a location for the resource. For example, if a GPU can render with a vertex buffer from either local or non-local memory with little difference in performance, that vertex buffer is low priority. Other more critical resources (for example, a render target or texture) can then occupy the faster memory.
<code>DXGI_RESOURCE_PRIORITY_NORMAL</code> (0x78000000)	The eviction priority of the resource is normal. The placement of the resource is important, but not critical, for performance. The resource is placed in its preferred location instead of a low-priority resource.
	The eviction priority of the resource is high. The

DXGI_RESOURCE_PRIORITY_HIGH (0xa0000000)	resource is placed in its preferred location instead of a low-priority or normal-priority resource.
DXGI_RESOURCE_PRIORITY_MAXIMUM (0xc8000000)	The resource is evicted from memory only if there is no other way of resolving the memory requirement.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

The eviction priority is a memory-management variable that is used by DXGI for determining how to populate overcommitted memory.

You can set priority levels other than the defined values when appropriate. For example, you can set a resource with a priority level of 0x78000001 to indicate that the resource is slightly above normal.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGIResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface interface (dxgi.h)

Article 05/25/2021

The **IDXGISurface** interface implements methods for image-data objects.

Inheritance

The **IDXGISurface** interface derives from [IDXGIDeviceSubObject](#).

Methods

The **IDXGISurface** interface has these methods.

IDXGISurface::GetDesc
Get a description of the surface.
IDXGISurface::Map
Get a pointer to the data contained in the surface, and deny GPU access to the surface.
IDXGISurface::Unmap
Invalidate the pointer to the surface retrieved by IDXGISurface::Map and re-enable GPU access to the resource.

Remarks

An image-data object is a 2D section of memory, commonly called a surface. To get the surface from an output, call [IDXGIOutput::GetDisplaySurfaceData](#).

Runtimes earlier than Direct3D 12 automatically create an **IDXGISurface** interface when they create a Direct3D resource object that represents a surface. **IDXGISurface** interfaces are not supported in Direct3D 12. For example, the runtime creates an **IDXGISurface** interface when you call [ID3D11Device::CreateTexture2D](#) or [ID3D10Device::CreateTexture2D](#) to create a 2D texture. To retrieve the **IDXGISurface** interface that represents the 2D texture surface, call [ID3D11Texture2D::QueryInterface](#) or [ID3D10Texture2D::QueryInterface](#). In this call, you must pass the identifier of **IDXGISurface**. If the 2D texture has only a single MIP-map level and does not consist of an array of textures, [QueryInterface](#) succeeds and returns a pointer to the **IDXGISurface**

interface pointer. Otherwise, `QueryInterface` fails and does not return the pointer to `IDXGISurface`.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIDeviceSubObject](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface::GetDesc method (dxgi.h)

Article 10/13/2021

Get a description of the surface.

Syntax

C++

```
HRESULT GetDesc(  
    [out] DXGI_SURFACE_DESC *pDesc  
) ;
```

Parameters

[out] pDesc

Type: [DXGI_SURFACE_DESC*](#)

A pointer to the surface description (see [DXGI_SURFACE_DESC](#)).

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface::Map method (dxgi.h)

Article 10/13/2021

Get a pointer to the data contained in the surface, and deny GPU access to the surface.

Syntax

C++

```
HRESULT Map(
    [out] DXGI_MAPPED_RECT *pLockedRect,
    UINT                 MapFlags
);
```

Parameters

[out] pLockedRect

Type: [DXGI_MAPPED_RECT*](#)

A pointer to the surface data (see [DXGI_MAPPED_RECT](#)).

MapFlags

Type: [UINT](#)

CPU read-write flags. These flags can be combined with a logical OR.

- [DXGI_MAP_READ](#) - Allow CPU read access.
- [DXGI_MAP_WRITE](#) - Allow CPU write access.
- [DXGI_MAP_DISCARD](#) - Discard the previous contents of a resource when it is mapped.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

Use [IDXGISurface::Map](#) to access a surface from the CPU. To release a mapped surface (and allow GPU access) call [IDXGISurface::Unmap](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISurface](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface::Unmap method (dxgi.h)

Article 06/29/2021

Invalidate the pointer to the surface retrieved by [IDXGISurface::Map](#) and re-enable GPU access to the resource.

Syntax

C++

```
HRESULT Unmap();
```

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISurface](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface1 interface (dxgi.h)

Article 08/23/2022

The [IDXGISurface1](#) interface extends the [IDXGISurface](#) by adding support for using Windows Graphics Device Interface (GDI) to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface.

Inheritance

The [IDXGISurface1](#) interface inherits from [IDXGISurface](#). [IDXGISurface1](#) also has these types of members:

Methods

The [IDXGISurface1](#) interface has these methods.

[IDXGISurface1::GetDC](#)

Returns a device context (DC) that allows you to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface using Windows Graphics Device Interface (GDI).

[IDXGISurface1::ReleaseDC](#)

Releases the GDI device context (DC) that is associated with the current surface and allows you to use Direct3D to render.

Remarks

This interface is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

An image-data object is a 2D section of memory, commonly called a surface. To get the surface from an output, call [IDXGIOutput::GetDisplaySurfaceData](#). Then, call [QueryInterface](#) on the [IDXGISurface](#) object that [IDXGIOutput::GetDisplaySurfaceData](#) returns to retrieve the [IDXGISurface1](#) interface.

Any object that supports [IDXGISurface](#) also supports [IDXGISurface1](#).

The runtime automatically creates an **IDXGISurface1** interface when it creates a Direct3D resource object that represents a surface. For example, the runtime creates an **IDXGISurface1** interface when you call **ID3D11Device::CreateTexture2D** or **ID3D10Device::CreateTexture2D** to create a 2D texture. To retrieve the **IDXGISurface1** interface that represents the 2D texture surface, call **ID3D11Texture2D::QueryInterface** or **ID3D10Texture2D::QueryInterface**. In this call, you must pass the identifier of **IDXGISurface1**. If the 2D texture has only a single MIP-map level and does not consist of an array of textures, **QueryInterface** succeeds and returns a pointer to the **IDXGISurface1** interface pointer. Otherwise, **QueryInterface** fails and does not return the pointer to **IDXGISurface1**.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGISurface](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface1::GetDC method (dxgi.h)

Article 08/23/2022

Returns a device context (DC) that allows you to render to a Microsoft DirectX Graphics Infrastructure (DXGI) surface using Windows Graphics Device Interface (GDI).

Syntax

C++

```
HRESULT GetDC(
    BOOL Discard,
    [out] HDC *phdc
);
```

Parameters

Discard

Type: [BOOL](#)

A Boolean value that specifies whether to preserve Direct3D contents in the GDI DC. **TRUE** directs the runtime not to preserve Direct3D contents in the GDI DC; that is, the runtime discards the Direct3D contents. **FALSE** guarantees that Direct3D contents are available in the GDI DC.

[out] phdc

Type: [HDC*](#)

A pointer to an [HDC](#) handle that represents the current device context for GDI rendering.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; otherwise, an error code.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

After you use the **GetDC** method to retrieve a DC, you can render to the DXGI surface by using GDI.

The **GetDC** method readies the surface for GDI rendering and allows inter-operation between DXGI and GDI technologies.

Keep the following in mind when using this method:

- You must create the surface by using the [D3D11_RESOURCE_MISC_GDI_COMPATIBLE](#) flag for a surface or by using the [DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE](#) flag for swap chains, otherwise this method fails.
- You must release the device and call the [IDXGISurface1::ReleaseDC](#) method before you issue any new Direct3D commands.
- This method fails if an outstanding DC has already been created by this method.
- The format for the surface or swap chain must be [DXGI_FORMAT_B8G8R8A8_UNORM_SRGB](#) or [DXGI_FORMAT_B8G8R8A8_UNORM](#).
- On **GetDC**, the render target in the output merger of the Direct3D pipeline is unbound from the surface. You must call the [ID3D11DeviceContext::OMSetRenderTargets](#) method on the device prior to Direct3D rendering after GDI rendering.
- Prior to resizing buffers you must release all outstanding DCs.

You can also call **GetDC** on the back buffer at index 0 of a swap chain by obtaining an [IDXGISurface1](#) from the swap chain. The following code illustrates the process.

```
IDXGISwapChain* g_pSwapChain = NULL;
IDXGISurface1* g_pSurface1 = NULL;
...
//Setup the device and and swapchain
g_pSwapChain->GetBuffer(0, __uuidof(IDXGISurface1), (void**) &g_pSurface1);
g_pSurface1->GetDC( FALSE, &g_hDC );
...
//Draw on the DC using GDI
...
//When finish drawing release the DC
g_pSurface1->ReleaseDC( NULL );
```

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGISurface1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface1::ReleaseDC method (dxgi.h)

Article 08/23/2022

Releases the GDI device context (DC) that is associated with the current surface and allows you to use Direct3D to render.

Syntax

C++

```
HRESULT ReleaseDC(  
    [in, optional] RECT *pDirtyRect  
);
```

Parameters

[in, optional] pDirtyRect

Type: [RECT*](#)

A pointer to a **RECT** structure that identifies the dirty region of the surface.

A dirty region is any part of the surface that you used for GDI rendering and that you want to preserve. This area is used as a performance hint to graphics subsystem in certain scenarios. Do not use this parameter to restrict rendering to the specified rectangular region. If you pass in **NULL**, **ReleaseDC** considers the whole surface as dirty. Otherwise, **ReleaseDC** uses the area specified by the **RECT** as a performance hint to indicate what areas have been manipulated by GDI rendering.

You can pass a pointer to an empty **RECT** structure (a rectangle with no position or area) if you didn't change any content.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

This method is not supported by DXGI 1.0, which shipped in Windows Vista and Windows Server 2008. DXGI 1.1 support is required, which is available on Windows 7, Windows Server 2008 R2, and as an update to Windows Vista with Service Pack 2 (SP2) ([KB 971644](#)) and Windows Server 2008 ([KB 971512](#)).

Use the **ReleaseDC** method to release the DC and indicate that your application finished all GDI rendering to this surface.

You must call the **ReleaseDC** method before you can use Direct3D to perform additional rendering.

Prior to resizing buffers you must release all outstanding DCs.

Requirements

Minimum supported client	Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGISurface1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain interface (dxgi.h)

Article 07/22/2021

An **IDXGISwapChain** interface implements one or more [surfaces](#) for storing rendered data before presenting it to an output.

Inheritance

The **IDXGISwapChain** interface inherits from [IDXGIDeviceSubObject](#). **IDXGISwapChain** also has these types of members:

Methods

The **IDXGISwapChain** interface has these methods.

IDXGISwapChain::GetBuffer
Accesses one of the swap-chain's back buffers.
IDXGISwapChain::GetContainingOutput
Get the output (the display monitor) that contains the majority of the client area of the target window.
IDXGISwapChain::GetDesc
Get a description of the swap chain.
IDXGISwapChain::GetFrameStatistics
Gets performance statistics about the last render frame.
IDXGISwapChain::GetFullscreenState
Get the state associated with full-screen mode.
IDXGISwapChain::GetLastPresentCount
Gets the number of times that <code>IDXGISwapChain::Present</code> or <code>IDXGISwapChain1::Present1</code> has been called.

[IDXGISwapChain::Present](#)

Presents a rendered image to the user.

[IDXGISwapChain::ResizeBuffers](#)

Changes the swap chain's back buffer size, format, and number of buffers. This should be called when the application window is resized.

[IDXGISwapChain::ResizeTarget](#)

Resizes the output target.

[IDXGISwapChain::SetFullscreenState](#)

Sets the display state to windowed or full screen.

Remarks

You can create a swap chain by calling [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#). You can also create a swap chain when you call [D3D11CreateDeviceAndSwapChain](#); however, you can then only access the subset of swap-chain functionality that the **IDXGISwapChain** interface provides.

Requirements

Target Platform	Windows
Header	dxgi.h

See also

[DXGI Interfaces](#)

[IDXGIDeviceSubObject](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGISwapChain::GetBuffer method (dxgi.h)

Article 10/13/2021

Accesses one of the swap-chain's back buffers.

Syntax

C++

```
HRESULT GetBuffer(
    UINT    Buffer,
    [in]  REFIID riid,
    [out] void   **ppSurface
);
```

Parameters

Buffer

Type: [UINT](#)

A zero-based buffer index.

If the swap chain's swap effect is [DXGI_SWAP_EFFECT_DISCARD](#), this method can only access the first buffer; for this situation, set the index to zero.

If the swap chain's swap effect is either [DXGI_SWAP_EFFECT_SEQUENTIAL](#) or [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#), only the swap chain's zero-index buffer can be read from and written to. The swap chain's buffers with indexes greater than zero can only be read from; so if you call the [IDXGIResource::GetUsage](#) method for such buffers, they have the [DXGI_USAGE_READ_ONLY](#) flag set.

[in] riid

Type: [REFIID](#)

The type of interface used to manipulate the buffer.

[out] ppSurface

Type: [void**](#)

A pointer to a back-buffer interface.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGISwapChain::GetContainingOutput method (dxgi.h)

Article 10/13/2021

Get the output (the display monitor) that contains the majority of the client area of the target window.

Syntax

C++

```
HRESULT GetContainingOutput(  
    [out] IDXGIOutput **ppOutput  
) ;
```

Parameters

[out] ppOutput

Type: [IDXGIOutput**](#)

A pointer to the output interface (see [IDXGIOutput](#)).

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

If the method succeeds, the output interface will be filled and its reference count incremented. When you are finished with it, be sure to release the interface to avoid a memory leak.

The output is also owned by the adapter on which the swap chain's device was created.

You cannot call [GetContainingOutput](#) on a swap chain that you created with [IDXGIFactory2::CreateSwapChainForComposition](#).

To determine the output corresponding to such a swap chain, you should call [IDXGIFactory::EnumAdapters](#) and then [IDXGIAdapter::EnumOutputs](#) to enumerate over all of the available outputs. You should then intersect the bounds of your [CoreWindow::Bounds](#) with the desktop coordinates of each output, as reported by [DXGI_OUTPUT_DESC1::DesktopCoordinates](#) or [DXGI_OUTPUT_DESC::DesktopCoordinates](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::GetDesc method (dxgi.h)

Article 10/13/2021

[Starting with Direct3D 11.1, we recommend not to use **GetDesc** anymore to get a description of the swap chain. Instead, use [IDXGISwapChain1::GetDesc1](#).]

Get a description of the swap chain.

Syntax

C++

```
HRESULT GetDesc(  
    [out] DXGI_SWAP_CHAIN_DESC *pDesc  
>;
```

Parameters

[out] pDesc

Type: [DXGI_SWAP_CHAIN_DESC*](#)

A pointer to the swap-chain description (see [DXGI_SWAP_CHAIN_DESC](#)).

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::GetFrameStatistics method (dxgi.h)

Article 10/13/2021

Gets performance statistics about the last render frame.

Syntax

C++

```
HRESULT GetFrameStatistics(  
    [out] DXGI_FRAME_STATISTICS *pStats  
);
```

Parameters

[out] pStats

Type: [DXGI_FRAME_STATISTICS*](#)

A pointer to a [DXGI_FRAME_STATISTICS](#) structure for the frame statistics.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

You cannot use **GetFrameStatistics** for swap chains that both use the bit-block transfer (bitblt) presentation model and draw in windowed mode.

You can only use **GetFrameStatistics** for swap chains that either use the flip presentation model or draw in full-screen mode. You set the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value in the **SwapEffect** member of the [DXGI_SWAP_CHAIN_DESC1](#) structure to specify that the swap chain uses the flip presentation model.

Statistics are not reliable in many multiple monitor scenarios, as well as scenarios where other fullscreen apps are running.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::GetFullscreenState method (dxgi.h)

Article 10/13/2021

Get the state associated with full-screen mode.

Syntax

C++

```
HRESULT GetFullscreenState(  
    [out, optional] BOOL      *pFullscreen,  
    [out, optional] IDXGIOutput **ppTarget  
) ;
```

Parameters

[out, optional] pFullscreen

Type: [BOOL*](#)

A pointer to a boolean whose value is either:

- **TRUE** if the swap chain is in full-screen mode
- **FALSE** if the swap chain is in windowed mode

[out, optional] ppTarget

Type: [IDXGIOutput**](#)

A pointer to the output target (see [IDXGIOutput](#)) when the mode is full screen; otherwise **NULL**.

Return value

Type: [HRESULT](#)

Returns one of the following [DXGI_ERROR](#).

Remarks

When the swap chain is in full-screen mode, a pointer to the target output will be returned and its reference count will be incremented.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::GetLastPresentCount method (dxgi.h)

Article 10/13/2021

Gets the number of times that [IDXGISwapChain::Present](#) or [IDXGISwapChain1::Present1](#) has been called.

Syntax

C++

```
HRESULT GetLastPresentCount(  
    [out] UINT *pLastPresentCount  
>;
```

Parameters

[out] pLastPresentCount

Type: [UINT*](#)

A pointer to a variable that receives the number of calls.

Return value

Type: [HRESULT](#)

Returns one of the [DXGI_ERROR](#) values.

Remarks

For info about presentation statistics for a frame, see [DXGI_FRAME_STATISTICS](#).

Requirements

Target Platform	Windows
-----------------	---------

Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::Present method (dxgi.h)

Article 09/23/2022

Presents a rendered image to the user.

Syntax

C++

```
HRESULT Present(
    UINT SyncInterval,
    UINT Flags
);
```

Parameters

SyncInterval

Type: [UINT](#)

An integer that specifies how to synchronize presentation of a frame with the vertical blank.

For the bit-block transfer (bitblt) model ([DXGI_SWAP_EFFECT_DISCARD](#) or [DXGI_SWAP_EFFECT_SEQUENTIAL](#)), values are:

- 0 - The presentation occurs immediately, there is no synchronization.
- 1 through 4 - Synchronize presentation after the *n*th vertical blank.

For the flip model ([DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#)), values are:

- 0 - Cancel the remaining time on the previously presented frame and discard this frame if a newer frame is queued.
- 1 through 4 - Synchronize presentation for at least *n* vertical blanks.

For an example that shows how sync-interval values affect a flip presentation queue, see Remarks.

If the update region straddles more than one output (each represented by [IDXGIOutput](#)), **Present** performs the synchronization to the output that contains the

largest sub-rectangle of the target window's client area.

Flags

Type: [UINT](#)

An integer value that contains swap-chain presentation options. These options are defined by the [DXGI_PRESENT](#) constants.

Return value

Type: [HRESULT](#)

Possible return values include: S_OK, DXGI_ERROR_DEVICE_RESET or DXGI_ERROR_DEVICE_REMOVED (see [DXGI_ERROR](#)), DXGI_STATUS_OCCLUDED (see [DXGI_STATUS](#)), or D3DDDIERR_DEVICEREMOVED.

Note The **Present** method can return either DXGI_ERROR_DEVICE_REMOVED or D3DDDIERR_DEVICEREMOVED if a video card has been physically removed from the computer, or a driver upgrade for the video card has occurred.

Remarks

Starting with Direct3D 11.1, consider using [IDXGISwapChain1::Present1](#) because you can then use dirty rectangles and the scroll rectangle in the swap chain presentation and as such use less memory bandwidth and as a result less system power. For more info about using dirty rectangles and the scroll rectangle in swap chain presentation, see [Using dirty rectangles and the scroll rectangle in swap chain presentation](#).

For the best performance when flipping swap-chain buffers in a full-screen application, see [Full-Screen Application Performance Hints](#).

Because calling **Present** might cause the render thread to wait on the message-pump thread, be careful when calling this method in an application that uses multiple threads. For more details, see [Multithreading Considerations](#).

Differences between Direct3D 9 and Direct3D 10:

Specifying [DXGI_PRESENT_TEST](#) in the *Flags* parameter is analogous to [IDirect3DDevice9::TestCooperativeLevel](#) in Direct3D 9.

For flip presentation model swap chains that you create with the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value set, a successful presentation unbinds back buffer 0 from the graphics pipeline, except for when you pass the [DXGI_PRESENT_DO_NOT_SEQUENCE](#) flag in the *Flags* parameter.

For info about how data values change when you present content to the screen, see [Converting data for the color space](#).

Flip presentation model queue

Suppose the following frames with sync-interval values are queued from oldest (A) to newest (E) before you call **Present**.

A: 3, B: 0, C: 0, D: 1, E: 0

When you call **Present**, the runtime shows frame A for only 1 vertical blank interval. The runtime terminates frame A early because of the sync interval 0 in frame B. Then the runtime shows frame D for 1 vertical blank interval, and then frame E until you submit a new presentation. The runtime discards frames B and C.

Variable refresh rate displays

It is a requirement of variable refresh rate displays that tearing is enabled. The [CheckFeatureSupport](#) method can be used to determine if this feature is available, and to set the required flags refer to the descriptions of [DXGI_PRESENT_ALLOW_TEARING](#) and [DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING](#), and [Variable refresh rate displays](#).

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::ResizeBuffers method (dxgi.h)

Article04/02/2021

Changes the swap chain's back buffer size, format, and number of buffers. This should be called when the application window is resized.

Syntax

C++

```
HRESULT ResizeBuffers(
    UINT      BufferCount,
    UINT      Width,
    UINT      Height,
    DXGI_FORMAT NewFormat,
    UINT      SwapChainFlags
);
```

Parameters

BufferCount

Type: [UINT](#)

The number of buffers in the swap chain (including all back and front buffers). This number can be different from the number of buffers with which you created the swap chain. This number can't be greater than [DXGI_MAX_SWAP_CHAIN_BUFFERS](#). Set this number to zero to preserve the existing number of buffers in the swap chain. You can't specify less than two buffers for the flip presentation model.

Width

Type: [UINT](#)

The new width of the back buffer. If you specify zero, DXGI will use the width of the client area of the target window. You can't specify the width as zero if you called the [IDXGIFactory2::CreateSwapChainForComposition](#) method to create the swap chain for a composition surface.

Height

Type: [UINT](#)

The new height of the back buffer. If you specify zero, DXGI will use the height of the client area of the target window. You can't specify the height as zero if you called the [IDXGIFactory2::CreateSwapChainForComposition](#) method to create the swap chain for a composition surface.

NewFormat

Type: [DXGI_FORMAT](#)

A [DXGI_FORMAT](#)-typed value for the new format of the back buffer. Set this value to [DXGI_FORMAT_UNKNOWN](#) to preserve the existing format of the back buffer. The flip presentation model supports a more restricted set of formats than the bit-block transfer (bitblt) model.

SwapChainFlags

Type: [UINT](#)

A combination of [DXGI_SWAP_CHAIN_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for swap-chain behavior.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

You can't resize a swap chain unless you release all outstanding references to its back buffers. You must release all of its direct and indirect references on the back buffers in order for [ResizeBuffers](#) to succeed.

Direct references are held by the application after it calls [AddRef](#) on a resource.

Indirect references are held by views to a resource, binding a view of the resource to a device context, a command list that used the resource, a command list that used a view to that resource, a command list that executed another command list that used the resource, and so on.

Before you call **ResizeBuffers**, ensure that the application releases all references (by calling the appropriate number of [Release](#) invocations) on the resources, any views to the resource, and any command lists that use either the resources or views, and ensure that neither the resource nor a view is still bound to a device context. You can use [ID3D11DeviceContext::ClearState](#) to ensure that all references are released. If a view is bound to a deferred context, you must discard the partially built command list as well (by calling [ID3D11DeviceContext::ClearState](#), then [ID3D11DeviceContext::FinishCommandList](#), then [Release](#) on the command list). After you call **ResizeBuffers**, you can re-query interfaces via [IDXGISwapChain::GetBuffer](#).

For swap chains that you created with [DXGI_SWAP_CHAIN_FLAG_GDI_COMPATIBLE](#), before you call **ResizeBuffers**, also call [IDXGISurface1::ReleaseDC](#) on the swap chain's back-buffer surface to ensure that you have no outstanding GDI device contexts (DCs) open.

We recommend that you call **ResizeBuffers** when a client window is resized (that is, when an application receives a WM_SIZE message).

The only difference between [IDXGISwapChain::ResizeBuffers](#) in Windows 8 versus Windows 7 is with flip presentation model swap chains that you create with the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) or [DXGI_SWAP_EFFECT_FLIP_DISCARD](#) value set. In Windows 8, you must call **ResizeBuffers** to realize a transition between full-screen mode and windowed mode; otherwise, your next call to the [IDXGISwapChain::Present](#) method fails.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGISwapChain::ResizeTarget method (dxgi.h)

Article 10/13/2021

Resizes the output target.

Syntax

C++

```
HRESULT ResizeTarget(
    [in] const DXGI_MODE_DESC *pNewTargetParameters
);
```

Parameters

[in] pNewTargetParameters

Type: [const DXGI_MODE_DESC*](#)

A pointer to a [DXGI_MODE_DESC](#) structure that describes the mode, which specifies the new width, height, format, and refresh rate of the target. If the format is [DXGI_FORMAT_UNKNOWN](#), **ResizeTarget** uses the existing format. We only recommend that you use [DXGI_FORMAT_UNKNOWN](#) when the swap chain is in full-screen mode as this method is not thread safe.

Return value

Type: [HRESULT](#)

Returns a code that indicates success or failure.

[DXGI_STATUS_MODE_CHANGE_IN_PROGRESS](#) is returned if a full-screen/windowed mode transition is occurring when this API is called. See [DXGI_ERROR](#) for additional DXGI error codes.

Remarks

ResizeTarget resizes the target window when the swap chain is in windowed mode, and changes the display mode on the target output when the swap chain is in full-screen

mode. Therefore, apps can call **ResizeTarget** to resize the target window (rather than a Microsoft Win32API such as [SetWindowPos](#)) without knowledge of the swap chain display mode.

If a Windows Store app calls **ResizeTarget**, it fails with [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

You cannot call **ResizeTarget** on a swap chain that you created with [IDXGIFactory2::CreateSwapChainForComposition](#).

Apps must still call [IDXGISwapChain::ResizeBuffers](#) after they call **ResizeTarget** because only **ResizeBuffers** can change the back buffers. But, if those apps have implemented window resize processing to call **ResizeBuffers**, they don't need to explicitly call **ResizeBuffers** after they call **ResizeTarget** because the window resize processing will achieve what the app requires.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain::SetFullscreenState method (dxgi.h)

Article 04/02/2021

Sets the display state to windowed or full screen.

Syntax

C++

```
HRESULT SetFullscreenState(  
    BOOL      Fullscreen,  
    IDXGIOOutput *pTarget  
>;
```

Parameters

`Fullscreen`

Type: **BOOL**

A Boolean value that specifies whether to set the display state to windowed or full screen. **TRUE** for full screen, and **FALSE** for windowed.

`pTarget`

Type: [in, optional] **IDXGIOOutput***

If you pass **TRUE** to the *Fullscreen* parameter to set the display state to full screen, you can optionally set this parameter to a pointer to an **IDXGIOOutput** interface for the output target that contains the swap chain. If you set this parameter to **NULL**, DXGI will choose the output based on the swap-chain's device and the output window's placement. If you pass **FALSE** to *Fullscreen*, then you must set this parameter to **NULL**.

Return value

Type: **HRESULT**

This method returns one of these values.

- **S_OK** if the action succeeded and the swap chain was placed in the requested state.
- **DXGI_ERROR_NOT_CURRENTLY_AVAILABLE** if the action failed. When this error is returned, your application can continue to run in windowed mode and try to switch to full-screen mode later. There are many reasons why a windowed-mode swap chain cannot switch to full-screen mode. Here are some examples.
 - The application is running over Terminal Server.
 - The output window is occluded.
 - The output window does not have keyboard focus.
 - Another application is already in full-screen mode.
- **DXGI_STATUS_MODE_CHANGE_IN_PROGRESS** is returned if a fullscreen/windowed mode transition is occurring when this API is called.
- Other error codes if you run out of memory or encounter another unexpected fault; these codes may be treated as hard, non-continuable errors.

Remarks

DXGI may change the display state of a swap chain in response to end user or system requests.

We recommend that you create a windowed swap chain and allow the end user to change the swap chain to full screen through [SetFullscreenState](#); that is, do not set the **Windowed** member of [DXGI_SWAP_CHAIN_DESC](#) to FALSE to force the swap chain to be full screen. However, if you create the swap chain as full screen, also provide the end user with a list of supported display modes because a swap chain that is created with an unsupported display mode might cause the display to go black and prevent the end user from seeing anything. Also, we recommend that you have a time-out confirmation screen or other fallback mechanism when you allow the end user to change display modes.

Notes for Windows Store apps

If a Windows Store app calls [SetFullscreenState](#) to set the display state to full screen, [SetFullscreenState](#) fails with [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#).

You cannot call [SetFullscreenState](#) on a swap chain that you created with [IDXGIFactory2::CreateSwapChainForComposition](#).

For the [flip presentation model](#), after you transition the display state to full screen, you must call [ResizeBuffers](#) to ensure that your call to [IDXGISwapChain1::Present1](#) succeeds.

Requirements

Target Platform	Windows
Header	dxgi.h
Library	DXGI.lib

See also

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgi1_2.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi1_2.h contains the following programming interfaces:

Interfaces

[IDXGIAdapter2](#)

The IDXGIAdapter2 interface represents a display subsystem, which includes one or more GPUs, DACs, and video memory.

[IDXGIDevice2](#)

The IDXGIDevice2 interface implements a derived class for DXGI objects that produce image data. The interface exposes methods to block CPU processing until the GPU completes processing, and to offer resources to the operating system.

[IDXGIDisplayControl](#)

The IDXGIDisplayControl interface exposes methods to indicate user preference for the operating system's stereoscopic 3D display behavior and to set stereoscopic 3D display status to enable or disable.

[IDXGIFactory2](#)

The IDXGIFactory2 interface includes methods to create a newer version swap chain with more features than IDXGISwapChain and to monitor stereoscopic 3D capabilities.

[IDXGIOutput1](#)

An IDXGIOutput1 interface represents an adapter output (such as a monitor).

[IDXGIOutputDuplication](#)

The IDXGIOutputDuplication interface accesses and manipulates the duplicated desktop image.

[IDXGIResource1](#)

An IDXGIResource1 interface extends the IDXGIResource interface by adding support for creating a subresource surface object and for creating a handle to a shared resource.

[IDXGISurface2](#)

The IDXGISurface2 interface extends the IDXGISurface1 interface by adding support for subresource surfaces and getting a handle to a shared resource.

[IDXGISwapChain1](#)

Provides presentation capabilities that are enhanced from IDXGISwapChain. These presentation capabilities consist of specifying dirty rectangles and scroll rectangle to optimize the presentation.

Structures

[DXGI_ADAPTER_DESC2](#)

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.2.

[DXGI_MODE_DESC1](#)

Describes a display mode and whether the display mode supports stereo.

[DXGI_OUTDUPL_DESC](#)

The DXGI_OUTDUPL_DESC structure describes the dimension of the output and the surface that contains the desktop image. The format of the desktop image is always DXGI_FORMAT_B8G8R8A8_UNORM.

[DXGI_OUTDUPL_FRAME_INFO](#)

The DXGI_OUTDUPL_FRAME_INFO structure describes the current desktop image.

[DXGI_OUTDUPL_MOVE_RECT](#)

The DXGI_OUTDUPL_MOVE_RECT structure describes the movement of a rectangle.

[DXGI_OUTDUPL_POINTER_POSITION](#)

The DXGI_OUTDUPL_POINTER_POSITION structure describes the position of the hardware cursor.

[DXGI_OUTDUPL_POINTER_SHAPE_INFO](#)

The DXGI_OUTDUPL_POINTER_SHAPE_INFO structure describes information about the cursor shape.

[DXGI_PRESENT_PARAMETERS](#)

Describes information about present that helps the operating system optimize presentation.

[DXGI_SWAP_CHAIN_DESC1](#)

Describes a swap chain. (DXGI_SWAP_CHAIN_DESC1)

[DXGI_SWAP_CHAIN_FULLSCREEN_DESC](#)

Describes full-screen mode for a swap chain.

Enumerations

[DXGI_ALPHA_MODE](#)

Identifies the alpha value, transparency behavior, of a surface.

[DXGI_COMPUTE_PREEMPTION_GRANULARITY](#)

Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current compute task.

[DXGI_GRAPHICS_PREEMPTION_GRANULARITY](#)

Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current graphics rendering task.

[DXGI_OFFER_RESOURCE_PRIORITY](#)

Identifies the importance of a resource's content when you call the IDXGIDevice2::OfferResources method to offer the resource.

[DXGI_OUTDUPL_POINTER_SHAPE_TYPE](#)

Identifies the type of pointer shape.

[DXGI_SCALING](#)

Identifies resize behavior when the back-buffer size does not match the size of the target output.

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_ADAPTER_DESC2 structure (dxgi1_2.h)

Article09/01/2022

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.2.

Syntax

C++

```
typedef struct DXGI_ADAPTER_DESC2 {
    WCHAR Description[128];
    UINT VendorId;
    UINT DeviceId;
    UINT SubSysId;
    UINT Revision;
    SIZE_T DedicatedVideoMemory;
    SIZE_T DedicatedSystemMemory;
    SIZE_T SharedSystemMemory;
    LUID AdapterLuid;
    UINT Flags;
    DXGI_GRAPHICS_PREEMPTION_GRANULARITY GraphicsPreemptionGranularity;
    DXGI_COMPUTE_PREEMPTION_GRANULARITY ComputePreemptionGranularity;
} DXGI_ADAPTER_DESC2;
```

Members

Description[128]

A string that contains the adapter description.

VendorId

The PCI ID of the hardware vendor.

DeviceId

The PCI ID of the hardware device.

SubSysId

The PCI ID of the sub system.

Revision

The PCI ID of the revision number of the adapter.

DedicatedVideoMemory

The number of bytes of dedicated video memory that are not shared with the CPU.

DedicatedSystemMemory

The number of bytes of dedicated system memory that are not shared with the CPU. This memory is allocated from available system memory at boot time.

SharedSystemMemory

The number of bytes of shared system memory. This is the maximum value of system memory that may be consumed by the adapter during operation. Any incidental memory consumed by the driver as it manages and uses video memory is additional.

AdapterLuid

A unique value that identifies the adapter. See [LUID](#) for a definition of the structure. LUID is defined in dxgi.h.

Flags

A value of the [DXGI_ADAPTER_FLAG](#) enumerated type that describes the adapter type. The [DXGI_ADAPTER_FLAG_REMOTE](#) flag is reserved.

GraphicsPreemptionGranularity

A value of the [DXGI_GRAPHICS_PREEMPTION_GRANULARITY](#) enumerated type that describes the granularity level at which the GPU can be preempted from performing its current graphics rendering task.

ComputePreemptionGranularity

A value of the [DXGI_COMPUTE_PREEMPTION_GRANULARITY](#) enumerated type that describes the granularity level at which the GPU can be preempted from performing its current compute task.

Remarks

The [DXGI_ADAPTER_DESC2](#) structure provides a DXGI 1.2 description of an adapter. This structure is initialized by using the [IDXGIAdapter2::GetDesc2](#) method.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

[IDXGIAAdapter2::GetDesc2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_ALPHA_MODE enumeration (dxgi1_2.h)

Article01/31/2022

Identifies the alpha value, transparency behavior, of a surface.

Syntax

C++

```
typedef enum DXGI_ALPHA_MODE {
    DXGI_ALPHA_MODE_UNSPECIFIED = 0,
    DXGI_ALPHA_MODE_PREMULTIPLIED = 1,
    DXGI_ALPHA_MODE_STRAIGHT = 2,
    DXGI_ALPHA_MODE_IGNORE = 3,
    DXGI_ALPHA_MODE_FORCE_DWORD = 0xffffffff
} ;
```

Constants

`DXGI_ALPHA_MODE_UNSPECIFIED`

Value: 0

Indicates that the transparency behavior is not specified.

`DXGI_ALPHA_MODE_PREMULTIPLIED`

Value: 1

Indicates that the transparency behavior is premultiplied. Each color is first scaled by the alpha value. The alpha value itself is the same in both straight and premultiplied alpha. Typically, no color channel value is greater than the alpha channel value. If a color channel value in a premultiplied format is greater than the alpha channel, the standard source-over blending math results in an additive blend.

`DXGI_ALPHA_MODE_STRAIGHT`

Value: 2

Indicates that the transparency behavior is not premultiplied. The alpha channel indicates the transparency of the color.

`DXGI_ALPHA_MODE_IGNORE`

Value: 3

Indicates to ignore the transparency behavior.

`DXGI_ALPHA_MODE_FORCE_DWORD`

Value: `0xffffffff`

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

Remarks

For more information about alpha mode, see [D2D1_ALPHA_MODE](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Enumerations](#)

[DXGI_SWAP_CHAIN_DESC1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_COMPUTE_PREEMPTION_GRANULARITY enumeration (dxgi1_2.h)

Article01/31/2022

Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current compute task.

Syntax

C++

```
typedef enum DXGI_COMPUTE_PREEMPTION_GRANULARITY {
    DXGI_COMPUTE_PREEMPTION_DMA_BUFFER_BOUNDARY = 0,
    DXGI_COMPUTE_PREEMPTION_DISPATCH_BOUNDARY = 1,
    DXGI_COMPUTE_PREEMPTION_THREAD_GROUP_BOUNDARY = 2,
    DXGI_COMPUTE_PREEMPTION_THREAD_BOUNDARY = 3,
    DXGI_COMPUTE_PREEMPTION_INSTRUCTION_BOUNDARY = 4
};
```

Constants

`DXGI_COMPUTE_PREEMPTION_DMA_BUFFER_BOUNDARY`

Value: 0

Indicates the preemption granularity as a compute packet.

`DXGI_COMPUTE_PREEMPTION_DISPATCH_BOUNDARY`

Value: 1

Indicates the preemption granularity as a dispatch (for example, a call to the [ID3D11DeviceContext::Dispatch](#) method). A dispatch is a part of a compute packet.

`DXGI_COMPUTE_PREEMPTION_THREAD_GROUP_BOUNDARY`

Value: 2

Indicates the preemption granularity as a thread group. A thread group is a part of a dispatch.

`DXGI_COMPUTE_PREEMPTION_THREAD_BOUNDARY`

Value: 3

Indicates the preemption granularity as a thread in a thread group. A thread is a part of a thread group.

DXGI_COMPUTE_PREEMPTION_INSTRUCTION_BOUNDARY

Value: 4

Indicates the preemption granularity as a compute instruction in a thread.

Remarks

You call the [IDXGIAAdapter2::GetDesc2](#) method to retrieve the granularity level at which the GPU can be preempted from performing its current compute task. The operating system specifies the compute granularity level in the **ComputePreemptionGranularity** member of the [DXGI_ADAPTER_DESC2](#) structure.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Enumerations](#)

[DXGI_ADAPTER_DESC2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_GRAPHICS_PREEMPTION_GRANULARITY enumeration (dxgi1_2.h)

Article01/31/2022

Identifies the granularity at which the graphics processing unit (GPU) can be preempted from performing its current graphics rendering task.

Syntax

C++

```
typedef enum DXGI_GRAPHICS_PREEMPTION_GRANULARITY {
    DXGI_GRAPHICS_PREEMPTION_DMA_BUFFER_BOUNDARY = 0,
    DXGI_GRAPHICS_PREEMPTION_PRIMITIVE_BOUNDARY = 1,
    DXGI_GRAPHICS_PREEMPTION_TRIANGLE_BOUNDARY = 2,
    DXGI_GRAPHICS_PREEMPTION_PIXEL_BOUNDARY = 3,
    DXGI_GRAPHICS_PREEMPTION_INSTRUCTION_BOUNDARY = 4
};
```

Constants

DXGI_GRAPHICS_PREEMPTION_DMA_BUFFER_BOUNDARY

Value: 0

Indicates the preemption granularity as a DMA buffer.

DXGI_GRAPHICS_PREEMPTION_PRIMITIVE_BOUNDARY

Value: 1

Indicates the preemption granularity as a graphics primitive. A primitive is a section in a DMA buffer and can be a group of triangles.

DXGI_GRAPHICS_PREEMPTION_TRIANGLE_BOUNDARY

Value: 2

Indicates the preemption granularity as a triangle. A triangle is a part of a primitive.

DXGI_GRAPHICS_PREEMPTION_PIXEL_BOUNDARY

Value: 3

Indicates the preemption granularity as a pixel. A pixel is a part of a triangle.

`DXGI_GRAPHICS_PREEMPTION_INSTRUCTION_BOUNDARY`

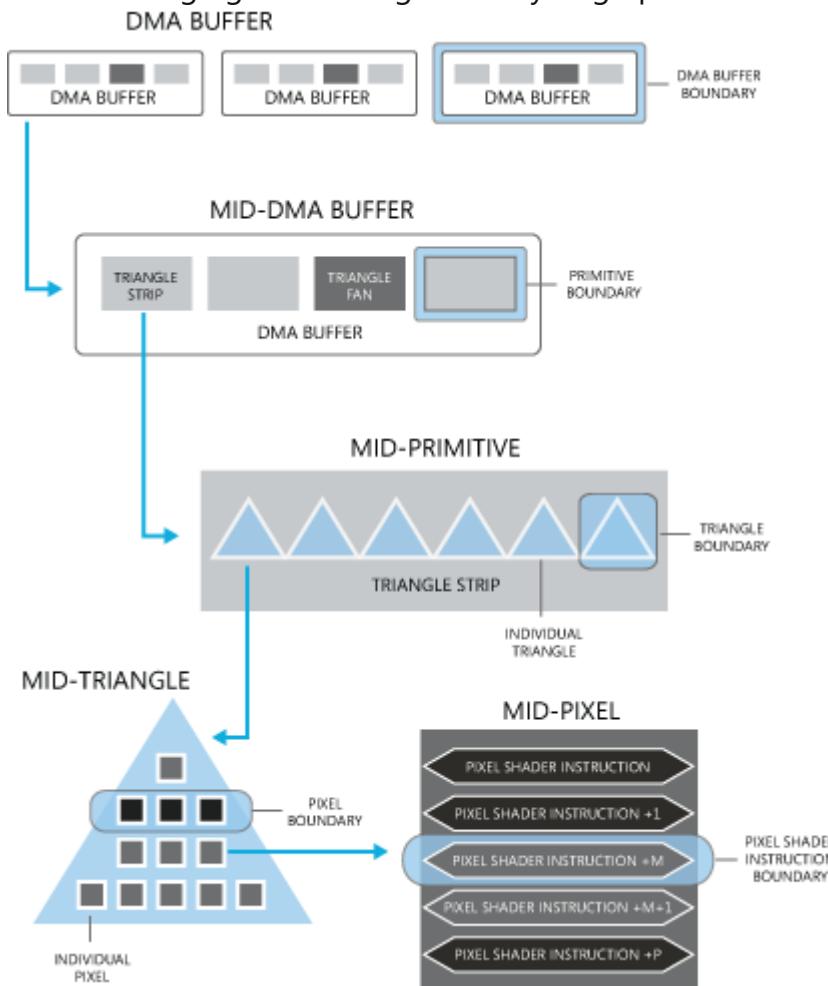
Value: 4

Indicates the preemption granularity as a graphics instruction. A graphics instruction operates on a pixel.

Remarks

You call the [IDXGIAAdapter2::GetDesc2](#) method to retrieve the granularity level at which the GPU can be preempted from performing its current graphics rendering task. The operating system specifies the graphics granularity level in the [GraphicsPreemptionGranularity](#) member of the [DXGI_ADAPTER_DESC2](#) structure.

The following figure shows granularity of graphics rendering tasks.



Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Enumerations](#)

[DXGI_ADAPTER_DESC2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_MODE_DESC1 structure (dxgi1_2.h)

Article 04/02/2021

Describes a display mode and whether the display mode supports stereo.

Syntax

C++

```
typedef struct DXGI_MODE_DESC1 {
    UINT             Width;
    UINT             Height;
    DXGI_RATIONAL   RefreshRate;
    DXGI_FORMAT      Format;
    DXGI_MODE_SCANLINE_ORDER ScanlineOrdering;
    DXGI_MODE_SCALING Scaling;
    BOOL            Stereo;
} DXGI_MODE_DESC1;
```

Members

Width

A value that describes the resolution width.

Height

A value that describes the resolution height.

RefreshRate

A [DXGI_RATIONAL](#) structure that describes the refresh rate in hertz.

Format

A [DXGI_FORMAT](#)-typed value that describes the display format.

ScanlineOrdering

A [DXGI_MODE_SCANLINE_ORDER](#)-typed value that describes the scan-line drawing mode.

Scaling

A [DXGI_MODE_SCALING](#)-typed value that describes the scaling mode.

Stereo

Specifies whether the full-screen display mode is stereo. **TRUE** if stereo; otherwise, **FALSE**.

Remarks

`DXGI_MODE_DESC1` is identical to [DXGI_MODE_DESC](#) except that `DXGI_MODE_DESC1` includes the **Stereo** member.

This structure is used by the [GetDisplayModeList1](#) and [FindClosestMatchingMode1](#) methods.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OFFER_RESOURCE_PRIORITY enumeration (dxgi1_2.h)

Article01/31/2022

Identifies the importance of a resource's content when you call the [IDXGIDevice2::OfferResources](#) method to offer the resource.

Syntax

C++

```
typedef enum _DXGI_OFFER_RESOURCE_PRIORITY {
    DXGI_OFFER_RESOURCE_PRIORITY_LOW = 1,
    DXGI_OFFER_RESOURCE_PRIORITY_NORMAL,
    DXGI_OFFER_RESOURCE_PRIORITY_HIGH
} DXGI_OFFER_RESOURCE_PRIORITY;
```

Constants

`DXGI_OFFER_RESOURCE_PRIORITY_LOW`

Value: 1

The resource is low priority. The operating system discards a low priority resource before other offered resources with higher priority. It is a good programming practice to mark a resource as low priority if it has no useful content.

`DXGI_OFFER_RESOURCE_PRIORITY_NORMAL`

The resource is normal priority. You mark a resource as normal priority if it has content that is easy to regenerate.

`DXGI_OFFER_RESOURCE_PRIORITY_HIGH`

The resource is high priority. The operating system discards other offered resources with lower priority before it discards a high priority resource. You mark a resource as high priority if it has useful content that is difficult to regenerate.

Remarks

Priority determines how likely the operating system is to discard an offered resource. Resources offered with lower priority are discarded first.

Requirements

Header	dxgi1_2.h
--------	-----------

See also

[DXGI Enumerations](#)

[IDXGIDevice2::OfferResources](#)

[IDXGIDevice2::ReclaimResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_DESC structure (dxgi1_2.h)

Article04/02/2021

The DXGI_OUTDUPL_DESC structure describes the dimension of the output and the surface that contains the desktop image. The format of the desktop image is always [DXGI_FORMAT_B8G8R8A8_UNORM](#).

Syntax

C++

```
typedef struct DXGI_OUTDUPL_DESC {
    DXGI_MODE_DESC ModeDesc;
    DXGI_MODE_ROTATION Rotation;
    BOOL DesktopImageInSystemMemory;
} DXGI_OUTDUPL_DESC;
```

Members

ModeDesc

A [DXGI_MODE_DESC](#) structure that describes the display mode of the duplicated output.

Rotation

A member of the [DXGI_MODE_ROTATION](#) enumerated type that describes how the duplicated output rotates an image.

DesktopImageInSystemMemory

Specifies whether the resource that contains the desktop image is already located in system memory. **TRUE** if the resource is in system memory; otherwise, **FALSE**. If this value is **TRUE** and the application requires CPU access, it can use the [IDXGIOutputDuplication::MapDesktopSurface](#) and [IDXGIOutputDuplication::UnMapDesktopSurface](#) methods to avoid copying the data into a staging buffer.

Remarks

This structure is used by [GetDesc](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_FRAME_INFO structure (dxgi1_2.h)

Article04/02/2021

The DXGI_OUTDUPL_FRAME_INFO structure describes the current desktop image.

Syntax

C++

```
typedef struct DXGI_OUTDUPL_FRAME_INFO {
    LARGE_INTEGER LastPresentTime;
    LARGE_INTEGER LastMouseUpdateTime;
    UINT AccumulatedFrames;
    BOOL RectsCoalesced;
    BOOL ProtectedContentMaskedOut;
    DXGI_OUTDUPL_POINTER_POSITION PointerPosition;
    UINT TotalMetadataBufferSize;
    UINT PointerShapeBufferSize;
} DXGI_OUTDUPL_FRAME_INFO;
```

Members

LastPresentTime

The time stamp of the last update of the desktop image. The operating system calls the [QueryPerformanceCounter](#) function to obtain the value. A zero value indicates that the desktop image was not updated since an application last called the [IDXGIOutputDuplication::AcquireNextFrame](#) method to acquire the next frame of the desktop image.

LastMouseUpdateTime

The time stamp of the last update to the mouse. The operating system calls the [QueryPerformanceCounter](#) function to obtain the value. A zero value indicates that the position or shape of the mouse was not updated since an application last called the [IDXGIOutputDuplication::AcquireNextFrame](#) method to acquire the next frame of the desktop image. The mouse position is always supplied for a mouse update. A new pointer shape is indicated by a non-zero value in the **PointerShapeBufferSize** member.

AccumulatedFrames

The number of frames that the operating system accumulated in the desktop image surface since the calling application processed the last desktop image. For more information about this number, see Remarks.

RectsCoalesced

Specifies whether the operating system accumulated updates by coalescing dirty regions. Therefore, the dirty regions might contain unmodified pixels. **TRUE** if dirty regions were accumulated; otherwise, **FALSE**.

ProtectedContentMaskedOut

Specifies whether the desktop image might contain protected content that was already blacked out in the desktop image. **TRUE** if protected content was already blacked; otherwise, **FALSE**. The application can use this information to notify the remote user that some of the desktop content might be protected and therefore not visible.

PointerPosition

A [DXGI_OUTDUPL_POINTER_POSITION](#) structure that describes the most recent mouse position if the **LastMouseUpdateTime** member is a non-zero value; otherwise, this value is ignored. This value provides the coordinates of the location where the top-left-hand corner of the pointer shape is drawn; this value is not the desktop position of the hot spot.

TotalMetadataBufferSize

Size in bytes of the buffers to store all the desktop update metadata for this frame. For more information about this size, see Remarks.

PointerShapeBufferSize

Size in bytes of the buffer to hold the new pixel data for the mouse shape. For more information about this size, see Remarks.

Remarks

A non-zero **LastMouseUpdateTime** indicates an update to either a mouse pointer position or a mouse pointer position and shape. That is, the mouse pointer position is always valid for a non-zero **LastMouseUpdateTime**; however, the application must check the value of the **PointerShapeBufferSize** member to determine whether the shape was updated too.

If only the pointer was updated (that is, the desktop image was not updated), the **AccumulatedFrames**, **TotalMetadataBufferSize**, and **LastPresentTime** members are set to zero.

An **AccumulatedFrames** value of one indicates that the application completed processing the last frame before a new desktop image was presented. If the **AccumulatedFrames** value is greater than one, more desktop image updates have occurred while the application processed the last desktop update. In this situation, the operating system accumulated the update regions. For more information about desktop updates, see Desktop Update Data.

A non-zero **TotalMetadataBufferSize** indicates the total size of the buffers that are required to store all the desktop update metadata. An application cannot determine the size of each type of metadata. The application must call the [IDXGIOutputDuplication::GetFrameDirtyRects](#), [IDXGIOutputDuplication::GetFrameMoveRects](#), or [IDXGIOutputDuplication::GetFramePointerShape](#) method to obtain information about each type of metadata.

Note To correct visual effects, an application must process the move region data before it processes the dirty rectangles.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

[IDXGIOutputDuplication::AcquireNextFrame](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_MOVE_RECT structure (dxgi1_2.h)

Article04/02/2021

The DXGI_OUTDUPL_MOVE_RECT structure describes the movement of a rectangle.

Syntax

C++

```
typedef struct DXGI_OUTDUPL_MOVE_RECT {  
    POINT SourcePoint;  
    RECT DestinationRect;  
} DXGI_OUTDUPL_MOVE_RECT;
```

Members

SourcePoint

The starting position of a rectangle.

DestinationRect

The target region to which to move a rectangle.

Remarks

This structure is used by [GetFrameMoveRects](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_POINTER_POSITION **structure (dxgi1_2.h)**

Article04/02/2021

The **DXGI_OUTDUPL_POINTER_POSITION** structure describes the position of the hardware cursor.

Syntax

C++

```
typedef struct DXGI_OUTDUPL_POINTER_POSITION {
    POINT Position;
    BOOL Visible;
} DXGI_OUTDUPL_POINTER_POSITION;
```

Members

Position

The position of the hardware cursor relative to the top-left of the adapter output.

Visible

Specifies whether the hardware cursor is visible. **TRUE** if visible; otherwise, **FALSE**. If the hardware cursor is not visible, the calling application does not display the cursor in the client.

Remarks

The **Position** member is valid only if the **Visible** member's value is set to **TRUE**.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
---------------------------------	---

Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

[DXGI_OUTDUPL_FRAME_INFO](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_POINTER_SHAPE_INFO structure (dxgi1_2.h)

Article04/02/2021

The **DXGI_OUTDUPL_POINTER_SHAPE_INFO** structure describes information about the cursor shape.

Syntax

C++

```
typedef struct DXGI_OUTDUPL_POINTER_SHAPE_INFO {
    UINT Type;
    UINT Width;
    UINT Height;
    UINT Pitch;
    POINT HotSpot;
} DXGI_OUTDUPL_POINTER_SHAPE_INFO;
```

Members

Type

A [DXGI_OUTDUPL_POINTER_SHAPE_TYPE](#)-typed value that specifies the type of cursor shape.

Width

The width in pixels of the mouse cursor.

Height

The height in scan lines of the mouse cursor.

Pitch

The width in bytes of the mouse cursor.

HotSpot

The position of the cursor's hot spot relative to its upper-left pixel. An application does not use the hot spot when it determines where to draw the cursor shape.

Remarks

An application draws the cursor shape with the top-left-hand corner drawn at the position that the **Position** member of the [DXGI_OUTDUPL_POINTER_POSITION](#) structure specifies; the application does not use the hot spot to draw the cursor shape.

An application calls the [IDXGIOutputDuplication::GetFramePointerShape](#) method to retrieve cursor shape information in a [DXGI_OUTDUPL_POINTER_SHAPE_INFO](#) structure.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

[IDXGIOutputDuplication::GetFramePointerShape](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTDUPL_POINTER_SHAPE_TYPE

enumeration (dxgi1_2.h)

Article01/31/2022

Identifies the type of pointer shape.

Syntax

C++

```
typedef enum DXGI_OUTDUPL_POINTER_SHAPE_TYPE {
    DXGI_OUTDUPL_POINTER_SHAPE_TYPE_MONOCHROME = 0x1,
    DXGI_OUTDUPL_POINTER_SHAPE_TYPE_COLOR = 0x2,
    DXGI_OUTDUPL_POINTER_SHAPE_TYPE_MASKED_COLOR = 0x4
};
```

Constants

`DXGI_OUTDUPL_POINTER_SHAPE_TYPE_MONOCHROME`

Value: `0x1`

The pointer type is a monochrome mouse pointer, which is a monochrome bitmap. The bitmap's size is specified by width and height in a 1 bits per pixel (bpp) device independent bitmap (DIB) format AND mask that is followed by another 1 bpp DIB format XOR mask of the same size.

`DXGI_OUTDUPL_POINTER_SHAPE_TYPE_COLOR`

Value: `0x2`

The pointer type is a color mouse pointer, which is a color bitmap. The bitmap's size is specified by width and height in a 32 bpp ARGB DIB format.

`DXGI_OUTDUPL_POINTER_SHAPE_TYPE_MASKED_COLOR`

Value: `0x4`

The pointer type is a masked color mouse pointer. A masked color mouse pointer is a 32 bpp ARGB format bitmap with the mask value in the alpha bits. The only allowed mask values are 0 and 0xFF. When the mask value is 0, the RGB value should replace the screen pixel. When the mask value is 0xFF, an XOR operation is performed on the RGB value and the screen pixel; the result replaces the screen pixel.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Enumerations](#)

[DXGI_OUTDUPL_POINTER_SHAPE_INFO](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_PRESENT_PARAMETERS structure (dxgi1_2.h)

Article10/06/2021

Describes information about present that helps the operating system optimize presentation.

Syntax

C++

```
typedef struct DXGI_PRESENT_PARAMETERS {
    UINT DirtyRectsCount;
    RECT *pDirtyRects;
    RECT *pScrollRect;
    POINT *pScrollOffset;
} DXGI_PRESENT_PARAMETERS;
```

Members

DirtyRectsCount

The number of updated rectangles that you update in the back buffer for the presented frame. The operating system uses this information to optimize presentation. You can set this member to 0 to indicate that you update the whole frame.

pDirtyRects

A list of updated rectangles that you update in the back buffer for the presented frame. An application must update every single pixel in each rectangle that it reports to the runtime; the application cannot assume that the pixels are saved from the previous frame. For more information about updating dirty rectangles, see Remarks. You can set this member to **NULL** if **DirtyRectsCount** is 0. An application must not update any pixel outside of the dirty rectangles.

pScrollRect

A pointer to the scrolled rectangle. The scrolled rectangle is the rectangle of the previous frame from which the runtime bit-block transfers (bitblts) content. The runtime also uses the scrolled rectangle to optimize presentation in terminal server and indirect display scenarios.

The scrolled rectangle also describes the destination rectangle, that is, the region on the current frame that is filled with scrolled content. You can set this member to **NULL** to indicate that no content is scrolled from the previous frame.

pScrollOffset

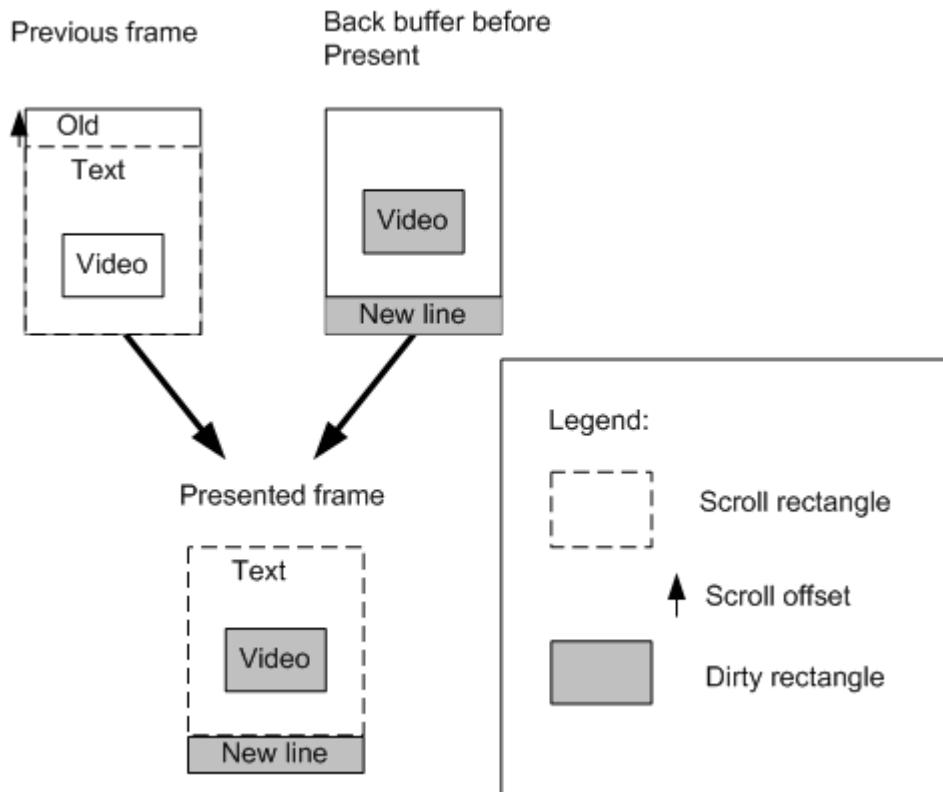
A pointer to the offset of the scrolled area that goes from the source rectangle (of previous frame) to the destination rectangle (of current frame). You can set this member to **NULL** to indicate no offset.

Remarks

This structure is used by the [Present1](#) method.

The scroll rectangle and the list of dirty rectangles could overlap. In this situation, the dirty rectangles take priority. Applications can then have pieces of dynamic content on top of a scrolled area. For example, an application could scroll a page and play video at the same time.

The following diagram and coordinates illustrate this example.



syntax

```
DirtyRectsCount = 2
pDirtyRects[ 0 ] = { 10, 30, 40, 50 } // Video
pDirtyRects[ 1 ] = { 0, 70, 50, 80 } // New line
*pScrollRect = { 0, 0, 50, 70 }
```

```
*pScrollOffset = { 0, -10 }
```

Parts of the previous frame and content that the application renders are combined to produce the final frame that the operating system presents on the display screen. Most of the window is scrolled from the previous frame. The application must update the video frame with the new chunk of content that appears due to scrolling.

The dashed rectangle shows the scroll rectangle in the current frame. The scroll rectangle is specified by the **pScrollRect** member. The arrow shows the scroll offset. The scroll offset is specified by the **pScrollOffset** member. Filled rectangles show dirty rectangles that the application updated with new content. The filled rectangles are specified by the **DirtyRectsCount** and **pDirtyRects** members.

The scroll rectangle and offset are not supported for the [DXGI_SWAP_EFFECT_DISCARD](#) or [DXGI_SWAP_EFFECT_SEQUENTIAL](#) present option. Dirty rectangles and scroll rectangle are not supported for multisampled swap chains.

The actual implementation of composition and necessary bitblts is different for the bitblt model and the flip model. For more info about these models, see [DXGI Flip Model](#).

For more info about the flip-model swap chain and optimizing presentation, see [Enhancing presentation with the flip model, dirty rectangles, and scrolled areas](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_SCALING enumeration (dxgi1_2.h)

Article01/31/2022

Identifies resize behavior when the back-buffer size does not match the size of the target output.

Syntax

C++

```
typedef enum DXGI_SCALING {
    DXGI_SCALING_STRETCH = 0,
    DXGI_SCALING_NONE = 1,
    DXGI_SCALING_ASPECT_RATIO_STRETCH = 2
};
```

Constants

`DXGI_SCALING_STRETCH`

Value: 0

Directs DXGI to make the back-buffer contents scale to fit the presentation target size. This is the implicit behavior of DXGI when you call the [IDXGIFactory::CreateSwapChain](#) method.

`DXGI_SCALING_NONE`

Value: 1

Directs DXGI to make the back-buffer contents appear without any scaling when the presentation target size is not equal to the back-buffer size. The top edges of the back buffer and presentation target are aligned together. If the `WS_EX_LAYOUTRTL` style is associated with the `HWND` handle to the target output window, the right edges of the back buffer and presentation target are aligned together; otherwise, the left edges are aligned together. All target area outside the back buffer is filled with window background color.

This value specifies that all target areas outside the back buffer of a swap chain are filled with the background color that you specify in a call to [IDXGISwapChain1::SetBackgroundColor](#).

`DXGI_SCALING_ASPECT_RATIO_STRETCH`

Value: 2

Directs DXGI to make the back-buffer contents scale to fit the presentation target size, while preserving the aspect ratio of the back-buffer. If the scaled back-buffer does not fill the presentation area, it will be centered with black borders.

This constant is supported on Windows Phone 8 and Windows 10.

Note that with legacy Win32 window swapchains, this works the same as `DXGI_SCALING_STRETCH`.

Remarks

The `DXGI_SCALING_NONE` value is supported only for flip presentation model swap chains that you create with the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value. You pass these values in a call to [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#).

`DXGI_SCALING_ASPECT_RATIO_STRETCH` will prefer to use a horizontal fill, otherwise it will use a vertical fill, using the following logic.

syntax

```
float aspectRatio = backBufferWidth / float(backBufferHeight);

// Horizontal fill
float scaledWidth = outputWidth;
float scaledHeight = outputWidth / aspectRatio;
if (scaledHeight >= outputHeight)
{
    // Do vertical fill
    scaledWidth = outputHeight * aspectRatio;
    scaledHeight = outputHeight;
}

float offsetX = (outputWidth - scaledWidth) * 0.5f;
float offsetY = (outputHeight - scaledHeight) * 0.5f;

rect.left = static_cast<LONG>(offsetX);
rect.top = static_cast<LONG>(offsetY);
rect.right = static_cast<LONG>(offsetX + scaledWidth);
rect.bottom = static_cast<LONG>(offsetY + scaledHeight);

rect.left = std::max(0, rect.left);
rect.top = std::max(0, rect.top);
```

```
rect.right = std::min<LONG>(static_cast<LONG>(outputWidth),  
rect.right);  
rect.bottom = std::min<LONG>(static_cast<LONG>(outputHeight),  
rect.bottom);
```

Note that *outputWidth* and *outputHeight* are the pixel sizes of the presentation target size. In the case of **CoreWindow**, this requires converting the *logicalWidth* and *logicalHeight* values from DIPS to pixels using the window's DPI property.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Enumerations](#)

[DXGI_SWAP_CHAIN_DESC1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SWAP_CHAIN_DESC1 structure (dxgi1_2.h)

Article07/27/2022

Describes a swap chain.

Syntax

C++

```
typedef struct DXGI_SWAP_CHAIN_DESC1 {
    UINT             Width;
    UINT             Height;
    DXGI_FORMAT      Format;
    BOOL            Stereo;
    DXGI_SAMPLE_DESC SampleDesc;
    DXGI_USAGE       BufferUsage;
    UINT            BufferCount;
    DXGI_SCALING     Scaling;
    DXGI_SWAP_EFFECT SwapEffect;
    DXGI_ALPHA_MODE  AlphaMode;
    UINT            Flags;
} DXGI_SWAP_CHAIN_DESC1;
```

Members

Width

A value that describes the resolution width. If you specify the width as zero when you call the [IDXGIFactory2::CreateSwapChainForHwnd](#) method to create a swap chain, the runtime obtains the width from the output window and assigns this width value to the swap-chain description. You can subsequently call the [IDXGISwapChain1::GetDesc1](#) method to retrieve the assigned width value. You cannot specify the width as zero when you call the [IDXGIFactory2::CreateSwapChainForComposition](#) method.

Height

A value that describes the resolution height. If you specify the height as zero when you call the [IDXGIFactory2::CreateSwapChainForHwnd](#) method to create a swap chain, the runtime obtains the height from the output window and assigns this height value to the swap-chain description. You can subsequently call the [IDXGISwapChain1::GetDesc1](#)

method to retrieve the assigned height value. You cannot specify the height as zero when you call the [IDXGIFactory2::CreateSwapChainForComposition](#) method.

Format

A [DXGI_FORMAT](#) structure that describes the display format.

Stereo

Specifies whether the full-screen display mode or the swap-chain back buffer is stereo. **TRUE** if stereo; otherwise, **FALSE**. If you specify stereo, you must also specify a flip-model swap chain (that is, a swap chain that has the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value set in the **SwapEffect** member).

SampleDesc

A [DXGI_SAMPLE_DESC](#) structure that describes multi-sampling parameters. This member is valid only with bit-block transfer (bitblt) model swap chains.

BufferUsage

A [DXGI_USAGE](#)-typed value that describes the surface usage and CPU access options for the back buffer. The back buffer can be used for shader input or render-target output.

BufferCount

A value that describes the number of buffers in the swap chain. When you create a full-screen swap chain, you typically include the front buffer in this value.

Scaling

A [DXGI_SCALING](#)-typed value that identifies resize behavior if the size of the back buffer is not equal to the target output.

SwapEffect

A [DXGI_SWAP_EFFECT](#)-typed value that describes the presentation model that is used by the swap chain and options for handling the contents of the presentation buffer after presenting a surface. You must specify the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value when you call the [IDXGIFactory2::CreateSwapChainForComposition](#) method because this method supports only [flip presentation model](#).

AlphaMode

A [DXGI_ALPHA_MODE](#)-typed value that identifies the transparency behavior of the swap-chain back buffer.

Flags

A combination of [DXGI_SWAP_CHAIN_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for swap-chain behavior.

Remarks

This structure is used by the [CreateSwapChainForHwnd](#), [CreateSwapChainForCoreWindow](#), [CreateSwapChainForComposition](#), [CreateSwapChainForCompositionSurfaceHandle](#), and [GetDesc1](#) methods.

Note You cannot cast a **DXGI_SWAP_CHAIN_DESC1** to a **DXGI_SWAP_CHAIN_DESC** and vice versa. An application must explicitly use the **IDXGISwapChain1::GetDesc1** method to retrieve the newer version of the swap-chain description structure.

In full-screen mode, there is a dedicated front buffer; in windowed mode, the desktop is the front buffer.

For a [flip-model](#) swap chain (that is, a swap chain that has the [DXGI_SWAP_EFFECT_FLIP_DISCARD](#) or [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value set in the **SwapEffect** member), you must set the **Format** member to [DXGI_FORMAT_R16G16B16A16_FLOAT](#), [DXGI_FORMAT_B8G8R8A8_UNORM](#), or [DXGI_FORMAT_R8G8B8A8_UNORM](#); you must set the **Count** member of the [DXGI_SAMPLE_DESC](#) structure that the **SampleDesc** member specifies to one and the **Quality** member of [DXGI_SAMPLE_DESC](#) to zero because multiple sample antialiasing (MSAA) is not supported; you must set the **BufferCount** member to from two to sixteen. For more info about flip-model swap chain, see DXGI Flip Model.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

[IDXGISwapChain1::GetDesc1](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

DXGI_SWAP_CHAIN_FULLSCREEN_DESC structure (dxgi1_2.h)

Article04/02/2021

Describes full-screen mode for a swap chain.

Syntax

C++

```
typedef struct DXGI_SWAP_CHAIN_FULLSCREEN_DESC {
    DXGI_RATIONAL RefreshRate;
    DXGI_MODE_SCANLINE_ORDER ScanlineOrdering;
    DXGI_MODE_SCALING Scaling;
    BOOL Windowed;
} DXGI_SWAP_CHAIN_FULLSCREEN_DESC;
```

Members

RefreshRate

A [DXGI_RATIONAL](#) structure that describes the refresh rate in hertz.

ScanlineOrdering

A member of the [DXGI_MODE_SCANLINE_ORDER](#) enumerated type that describes the scan-line drawing mode.

Scaling

A member of the [DXGI_MODE_SCALING](#) enumerated type that describes the scaling mode.

Windowed

A Boolean value that specifies whether the swap chain is in windowed mode. **TRUE** if the swap chain is in windowed mode; otherwise, **FALSE**.

Remarks

This structure is used by the [CreateSwapChainForHwnd](#) and [GetFullscreenDesc](#) methods.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Header	dxgi1_2.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter2 interface (dxgi1_2.h)

Article07/22/2021

The [IDXGIAdapter2](#) interface represents a display subsystem, which includes one or more GPUs, DACs, and video memory.

Inheritance

The [IDXGIAdapter2](#) interface inherits from [IDXGIAdapter1](#). [IDXGIAdapter2](#) also has these types of members:

Methods

The [IDXGIAdapter2](#) interface has these methods.

[IDXGIAdapter2::GetDesc2](#)

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.2 description of an adapter or video card.

Remarks

A display subsystem is often referred to as a video card; however, on some computers, the display subsystem is part of the motherboard.

To enumerate the display subsystems, use [IDXGIFactory1::EnumAdapters1](#).

To get an interface to the adapter for a particular device, use [IDXGIDevice::GetAdapter](#).

To create a software adapter, use [IDXGIFactory::CreateSoftwareAdapter](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
---------------------------------	---

Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIAAdapter1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter2::GetDesc2 method (dxgi1_2.h)

Article 10/13/2021

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.2 description of an adapter or video card. This description includes information about the granularity at which the graphics processing unit (GPU) can be preempted from performing its current task.

Syntax

C++

```
HRESULT GetDesc2(  
    [out] DXGI_ADAPTER_DESC2 *pDesc  
);
```

Parameters

[out] pDesc

A pointer to a [DXGI_ADAPTER_DESC2](#) structure that describes the adapter.

This parameter must not be **NULL**. On **feature level 9** graphics hardware, earlier versions of **GetDesc2** ([GetDesc](#) and [GetDesc1](#)) return zeros for the PCI ID in the **VendorId**, **DeviceId**, **SubSysId**, and **Revision** members of the adapter description structure and "Software Adapter" for the description string in the **Description** member. **GetDesc2** returns the actual feature level 9 hardware values in these members.

Return value

Returns **S_OK** if successful; otherwise, returns **E_INVALIDARG** if the *pDesc* parameter is **NULL**.

Remarks

Use the **GetDesc2** method to get a DXGI 1.2 description of an adapter. To get a DXGI 1.1 description, use the [IDXGIAdapter1::GetDesc1](#) method. To get a DXGI 1.0 description, use the [IDXGIAdapter::GetDesc](#) method.

The Windows Display Driver Model (WDDM) scheduler can preempt the GPU's execution of application tasks. The granularity at which the GPU can be preempted from performing its current task in the WDDM 1.1 or earlier driver model is a direct memory access (DMA) buffer for graphics tasks or a compute packet for compute tasks. The GPU can switch between tasks only after it completes the currently executing unit of work, a DMA buffer or a compute packet.

A DMA buffer is the largest independent unit of graphics work that the WDDM scheduler can submit to the GPU. This buffer contains a set of GPU instructions that the WDDM driver and GPU use. A compute packet is the largest independent unit of compute work that the WDDM scheduler can submit to the GPU. A compute packet contains dispatches (for example, calls to the [ID3D11DeviceContext::Dispatch](#) method), which contain thread groups. The WDDM 1.2 or later driver model allows the GPU to be preempted at finer granularity levels than a DMA buffer or compute packet. You can use the [GetDesc2](#) method to retrieve the granularity levels for graphics and compute tasks.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIAdapter2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice2 interface (dxgi1_2.h)

Article07/27/2022

The **IDXGIDevice2** interface implements a derived class for DXGI objects that produce image data. The interface exposes methods to block CPU processing until the GPU completes processing, and to offer resources to the operating system.

Inheritance

The **IDXGIDevice2** interface inherits from [IDXGIDevice1](#). **IDXGIDevice2** also has these types of members:

Methods

The **IDXGIDevice2** interface has these methods.

IDXGIDevice2::EnqueueSetEvent
Flushes any outstanding rendering commands and sets the specified event object to the signaled state after all previously submitted rendering commands complete.
IDXGIDevice2::OfferResources
Allows the operating system to free the video memory of resources by discarding their content. (<code>IDXGIDevice2.OfferResources</code>)
IDXGIDevice2::ReclaimResources
Restores access to resources that were previously offered by calling <code>IDXGIDevice2::OfferResources</code> .

Remarks

The **IDXGIDevice2** interface is designed for use by DXGI objects that need access to other DXGI objects. This interface is useful to applications that do not use Direct3D to communicate with DXGI.

The Direct3D create device functions return a Direct3D device object. This Direct3D device object implements the [IUnknown](#) interface. You can query this Direct3D device object for the device's corresponding **IDXGIDevice2** interface. To retrieve the **IDXGIDevice2** interface of a Direct3D device, use the following code:

```
IDXGIDevice2 * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice2), (void**)&pDXGIDevice);
```

Windows Phone 8: This API is supported.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIDevice1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice2::EnqueueSetEvent method (dxgi1_2.h)

Article 10/13/2021

Flushes any outstanding rendering commands and sets the specified event object to the signaled state after all previously submitted rendering commands complete.

Syntax

C++

```
HRESULT EnqueueSetEvent(  
    [in] HANDLE hEvent  
);
```

Parameters

[in] hEvent

A handle to the event object. The [CreateEvent](#) or [OpenEvent](#) function returns this handle. All types of event objects (manual-reset, auto-reset, and so on) are supported.

The handle must have the EVENT_MODIFY_STATE access right. For more information about access rights, see [Synchronization Object Security and Access Rights](#).

Return value

Returns S_OK if successful; otherwise, returns one of the following values:

- E_OUTOFMEMORY if insufficient memory is available to complete the operation.
- E_INVALIDARG if the parameter was validated and determined to be incorrect.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **EnqueueSetEvent** fails with E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

EnqueueSetEvent calls the **SetEvent** function on the event object after all previously submitted rendering commands complete or the device is removed.

After an application calls **EnqueueSetEvent**, it can immediately call the **WaitForSingleObject** function to put itself to sleep until rendering commands complete.

You cannot use **EnqueueSetEvent** to determine work completion that is associated with presentation (**IDXGISwapChain::Present**); instead, we recommend that you use **IDXGISwapChain::GetFrameStatistics**.

Examples

The following example code shows how to use **EnqueueSetEvent**.

```
void BlockingFinish( IDXGIDevice2* pDevice )
{
    // Create a manual-reset event object.
    hEvent = CreateEvent(
        NULL,                  // default security attributes
        TRUE,                 // manual-reset event
        FALSE,                // initial state is nonsignaled
        FALSE
    );

    if (hEvent == NULL)
    {
        printf("CreateEvent failed (%d)\n", GetLastError());
        return;
    }

    pDevice->EnqueueSetEvent(hEvent);

    DWORD dwWaitResult = WaitForSingleObject(
        hEvent, // event handle
        INFINITE); // indefinite wait

    switch (dwWaitResult)
    {
        // Event object was signaled
        case WAIT_OBJECT_0:
            // Commands completed
            break;

        // An error occurred
        default:
            printf("Wait error (%d)\n", GetLastError());
            return 0;
    }
}
```

```
        CloseHandle(hEvent);
    }
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIDevice2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice2::OfferResources method (dxgi1_2.h)

Article 07/27/2022

Allows the operating system to free the video memory of resources by discarding their content.

Syntax

C++

```
HRESULT OfferResources(
    [in] UINT NumResources,
    [in] IDXGIResource * const *ppResources,
    [in] DXGI_OFFER_RESOURCE_PRIORITY Priority
);
```

Parameters

[in] NumResources

The number of resources in the *ppResources* argument array.

[in] ppResources

An array of pointers to [IDXGIResource](#) interfaces for the resources to offer.

[in] Priority

A [DXGI_OFFER_RESOURCE_PRIORITY](#)-typed value that indicates how valuable data is.

Return value

OfferResources returns:

- S_OK if resources were successfully offered
- E_INVALIDARG if a resource in the array or the priority is invalid

Remarks

The priority value that the *Priority* parameter specifies describes how valuable the caller considers the content to be. The operating system uses the priority value to discard resources in order of priority. The operating system discards a resource that is offered with low priority before it discards a resource that is offered with a higher priority.

If you call **OfferResources** to offer a resource while the resource is bound to the pipeline, the resource is unbound. You cannot call **OfferResources** on a resource that is mapped. After you offer a resource, the resource cannot be mapped or bound to the pipeline until you call the [IDXGIDevice2::ReclaimResource](#) method to reclaim the resource. You cannot call **OfferResources** to offer immutable resources.

To offer shared resources, call **OfferResources** on only one of the sharing devices. To ensure exclusive access to the resources, you must use an [IDXGIAKeyedMutex](#) object and then call **OfferResources** only while you hold the mutex. In fact, you can't offer shared resources unless you use [IDXGIAKeyedMutex](#) because offering shared resources without using [IDXGIAKeyedMutex](#) isn't supported.

Note The user mode display driver might not immediately offer the resources that you specified in a call to **OfferResources**. The driver can postpone offering them until the next call to [IDXGISwapChain::Present](#), [IDXGISwapChain1::Present1](#), or [ID3D11DeviceContext::Flush](#).

Platform Update for Windows 7: The runtime validates that **OfferResources** is used correctly on non-shared resources but doesn't perform the intended functionality. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIDevice2](#)

[IDXGIDevice2::ReclaimResource](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDevice2::ReclaimResources method (dxgi1_2.h)

Article 10/13/2021

Restores access to resources that were previously offered by calling [IDXGIDevice2::OfferResources](#).

Syntax

C++

```
HRESULT ReclaimResources(
    [in]          UINT      NumResources,
    [in]          IDXGIResource * const *ppResources,
    [out, optional] BOOL     *pDiscarded
);
```

Parameters

[in] NumResources

The number of resources in the *ppResources* argument and *pDiscarded* argument arrays.

[in] ppResources

An array of pointers to [IDXGIResource](#) interfaces for the resources to reclaim.

[out, optional] pDiscarded

A pointer to an array that receives Boolean values. Each value in the array corresponds to a resource at the same index that the *ppResources* parameter specifies. The runtime sets each Boolean value to TRUE if the corresponding resource's content was discarded and is now undefined, or to FALSE if the corresponding resource's old content is still intact. The caller can pass in **NULL**, if the caller intends to fill the resources with new content regardless of whether the old content was discarded.

Return value

ReclaimResources returns:

- S_OK if resources were successfully reclaimed
- E_INVALIDARG if the resources are invalid

Remarks

After you call [IDXGIDevice2::OfferResources](#) to offer one or more resources, you must call **ReclaimResources** before you can use those resources again. You must check the values in the array at *pDiscarded* to determine whether each resource's content was discarded. If a resource's content was discarded while it was offered, its current content is undefined. Therefore, you must overwrite the resource's content before you use the resource.

To reclaim shared resources, call **ReclaimResources** only on one of the sharing devices. To ensure exclusive access to the resources, you must use an [IDXGIKeyedMutex](#) object and then call **ReclaimResources** only while you hold the mutex.

Platform Update for Windows 7: The runtime validates that **ReclaimResources** is used correctly on non-shared resources but doesn't perform the intended functionality. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIDevice2](#)

[IDXGIDevice2::OfferResources](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDisplayControl interface (dxgi1_2.h)

Article 07/22/2021

The **IDXGIDisplayControl** interface exposes methods to indicate user preference for the operating system's stereoscopic 3D display behavior and to set stereoscopic 3D display status to enable or disable.

We recommend that you not use **IDXGIDisplayControl** to query or set system-wide stereoscopic 3D settings in your stereoscopic 3D apps. Instead, for your windowed apps, call the [IDXGIFactory2::IsWindowedStereoEnabled](#) method to determine whether to render in stereo; for your full-screen apps, call the [IDXGIOutput1::GetDisplayModeList1](#) method and then determine whether any of the returned display modes support rendering in stereo.

Inheritance

The **IDXGIDisplayControl** interface inherits from the [IUnknown](#) interface.

IDXGIDisplayControl also has these types of members:

Methods

The **IDXGIDisplayControl** interface has these methods.

[IDXGIDisplayControl::IsStereoEnabled](#)

Retrieves a Boolean value that indicates whether the operating system's stereoscopic 3D display behavior is enabled.

[IDXGIDisplayControl::SetStereoEnabled](#)

Set a Boolean value to either enable or disable the operating system's stereoscopic 3D display behavior.

Remarks

Note The **IDXGIDisplayControl** interface is only used by the **Display** app of the operating system's Control Panel or by control applets from third party graphics vendors. This interface is not meant for developers of end-user apps.

Note The **IDXGIDisplayControl** interface does not exist for Windows Store apps.

Call [QueryInterface](#) from a factory object ([IDXGIFactory](#), [IDXGIFactory1](#) or [IDXGIFactory2](#)) to retrieve the **IDXGIDisplayControl** interface. The following code shows how.

```
IDXGIDisplayControl * pDXGIDisplayControl;  
hr = g_pDXGIFactory->QueryInterface(__uuidof(IDXGIDisplayControl), (void **)&pDXGIDisplayControl);
```

The operating system processes changes to stereo-enabled configuration asynchronously. Therefore, these changes might not be immediately visible in every process that calls [IDXGIDisplayControl::IsStereoEnabled](#) to query for stereo configuration. Control applets can use the [IDXGIFactory2::RegisterStereoStatusEvent](#) or [IDXGIFactory2::RegisterStereoStatusWindow](#) method to register for notifications of all stereo configuration changes.

Platform Update for Windows 7: Stereoscopic 3D display behavior isn't available with the Platform Update for Windows 7. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDisplayControl::IsStereoEnabled method (dxgi1_2.h)

Article 06/29/2021

Retrieves a Boolean value that indicates whether the operating system's stereoscopic 3D display behavior is enabled.

Syntax

C++

```
BOOL IsStereoEnabled();
```

Return value

IsStereoEnabled returns TRUE when the operating system's stereoscopic 3D display behavior is enabled and FALSE when this behavior is disabled.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **IsStereoEnabled** always returns FALSE because stereoscopic 3D display behavior isn't available with the Platform Update for Windows 7. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

You pass a Boolean value to the [IDXGIDisplayControl::SetStereoEnabled](#) method to either enable or disable the operating system's stereoscopic 3D display behavior. TRUE enables the operating system's stereoscopic 3D display behavior and FALSE disables it.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
---------------------------------	---

Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIDisplayControl](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDisplayControl::SetStereoEnabled method (dxgi1_2.h)

Article04/02/2021

Set a Boolean value to either enable or disable the operating system's stereoscopic 3D display behavior.

Syntax

C++

```
void SetStereoEnabled(  
    BOOL enabled  
)
```

Parameters

`enabled`

A Boolean value that either enables or disables the operating system's stereoscopic 3D display behavior. TRUE enables the operating system's stereoscopic 3D display behavior and FALSE disables it.

Return value

None

Remarks

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **SetStereoEnabled** doesn't change stereoscopic 3D display behavior because stereoscopic 3D display behavior isn't available with the Platform Update for Windows 7. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIDisplayControl](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2 interface (dxgi1_2.h)

Article07/22/2021

The **IDXGIFactory2** interface includes methods to create a newer version swap chain with more features than [IDXGISwapChain](#) and to monitor stereoscopic 3D capabilities.

Inheritance

The **IDXGIFactory2** interface inherits from [IDXGIFactory1](#). **IDXGIFactory2** also has these types of members:

Methods

The **IDXGIFactory2** interface has these methods.

IDXGIFactory2::CreateSwapChainForComposition
Creates a swap chain that you can use to send Direct3D content into the DirectComposition API or a Xaml framework to compose in a window.
IDXGIFactory2::CreateSwapChainForCoreWindow
Creates a swap chain that is associated with the CoreWindow object for the output window for the swap chain.
IDXGIFactory2::CreateSwapChainForHwnd
Creates a swap chain that is associated with an HWND handle to the output window for the swap chain.
IDXGIFactory2::GetSharedResourceAdapterLuid
Identifies the adapter on which a shared resource object was created.
IDXGIFactory2::IsWindowedStereoEnabled
Determines whether to use stereo mode.
IDXGIFactory2::RegisterOcclusionStatusEvent
Registers to receive notification of changes in occlusion status by using event signaling.

[IDXGIFactory2::RegisterOcclusionStatusWindow](#)

Registers an application window to receive notification messages of changes of occlusion status.

[IDXGIFactory2::RegisterStereoStatusEvent](#)

Registers to receive notification of changes in stereo status by using event signaling.

[IDXGIFactory2::RegisterStereoStatusWindow](#)

Registers an application window to receive notification messages of changes of stereo status.

[IDXGIFactory2::UnregisterOcclusionStatus](#)

Unregisters a window or an event to stop it from receiving notification when occlusion status changes.

[IDXGIFactory2::UnregisterStereoStatus](#)

Unregisters a window or an event to stop it from receiving notification when stereo status changes.

Remarks

To create a Microsoft DirectX Graphics Infrastructure (DXGI) 1.2 factory interface, pass [IDXGIFactory2](#) into either the [CreateDXGIFactory](#) or [CreateDXGIFactory1](#) function or call [QueryInterface](#) from a factory object that either [CreateDXGIFactory](#) or [CreateDXGIFactory1](#) returns.

Because you can create a Direct3D device without creating a swap chain, you might need to retrieve the factory that is used to create the device in order to create a swap chain. You can request the [IDXGIDevice](#), [IDXGIDevice1](#), or [IDXGIDevice2](#) interface from the Direct3D device and then use the [IDXGIOBJECT::GetParent](#) method to locate the factory. The following code shows how.

```
IDXGIDevice2 * pDXGIDevice;
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice2), (void **)&pDXGIDevice);

IDXGIAdapter * pDXGIAdapter;
hr = pDXGIDevice->GetParent(__uuidof(IDXGIAdapter), (void **)&pDXGIAdapter);

IDXGIFactory2 * pIDXGIFactory;
```

```
pDXGIAAdapter->GetParent(__uuidof(IDXGIFactory2), (void **)&pIDXGIFactory);
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::CreateSwapChainForComposition method (dxgi1_2.h)

Article02/16/2023

Creates a swap chain that you can use to send Direct3D content into the [DirectComposition API](#), to the [Windows.UI.Xaml](#) framework, or to [Windows UI Library \(WinUI\) XAML](#), to compose in a window.

Syntax

C++

```
HRESULT CreateSwapChainForComposition(
    [in]          IUnknown* pDevice,
    [in]          const DXGI_SWAP_CHAIN_DESC1* pDesc,
    [in, optional] IDXGIOutput* pRestrictToOutput,
    [out]         IDXGISwapChain1** ppSwapChain
);
```

Parameters

[in] pDevice

For Direct3D 11, and earlier versions of Direct3D, this is a pointer to the Direct3D device for the swap chain. For Direct3D 12 this is a pointer to a direct command queue (refer to [ID3D12CommandQueue](#)). This parameter cannot be **NULL**. Software drivers, like [D3D_DRIVER_TYPE_REFERENCE](#), are not supported for composition swap chains.

[in] pDesc

A pointer to a [DXGI_SWAP_CHAIN_DESC1](#) structure for the swap-chain description. This parameter cannot be **NULL**.

You must specify the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value in the **SwapEffect** member of [DXGI_SWAP_CHAIN_DESC1](#) because [CreateSwapChainForComposition](#) supports only [flip](#) presentation model.

You must also specify the [DXGI_SCALING_STRETCH](#) value in the **Scaling** member of [DXGI_SWAP_CHAIN_DESC1](#).

[in, optional] pRestrictToOutput

A pointer to the [IDXGIOOutput](#) interface for the output to restrict content to. You must also pass the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag in a [IDXGISwapChain1::Present1](#) call to force the content to appear blacked out on any other output. If you want to restrict the content to a different output, you must create a new swap chain. However, you can conditionally restrict content based on the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag.

Set this parameter to **NULL** if you don't want to restrict content to an output target.

[out] *ppSwapChain*

A pointer to a variable that receives a pointer to the [IDXGISwapChain1](#) interface for the swap chain that [CreateSwapChainForComposition](#) creates.

Return value

[CreateSwapChainForComposition](#) returns:

- **S_OK** if it successfully created a swap chain.
- **E_OUTOFMEMORY** if memory is unavailable to complete the operation.
- [DXGI_ERROR_INVALID_CALL](#) if the calling application provided invalid data, for example, if *pDesc* or *ppSwapChain* is **NULL**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic that are defined by the type of device that you pass to *pDevice*.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, [CreateSwapChainForComposition](#) fails with **E_NOTIMPL**. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

You can use composition swap chains with either:

- DirectComposition's [IDCompositionVisual](#) interface,
- System XAML's [SwapChainPanel](#) or [SwapChainBackgroundPanel](#) classes.
- Windows UI Library (WinUI) 3 XAML's [SwapChainPanel](#) or [SwapChainBackgroundPanel](#) classes.

For DirectComposition, you can call the [IDCompositionVisual::SetContent](#) method to set the swap chain as the content of a [visual object](#), which then allows you to bind the swap chain to the visual tree. For XAML, the [SwapChainBackgroundPanel](#) class exposes a

classic COM interface [ISwapChainBackgroundPanelNative](#). You can use the [ISwapChainBackgroundPanelNative::SetSwapChain](#) method to bind to the XAML UI graph. For info about how to use composition swap chains with XAML's [SwapChainBackgroundPanel](#) class, see [DirectX and XAML interop](#).

The [IDXGISwapChain::SetFullscreenState](#), [IDXGISwapChain::ResizeTarget](#), [IDXGISwapChain::GetContainingOutput](#), [IDXGISwapChain1::GetHwnd](#), and [IDXGISwapChain::GetCoreWindow](#) methods aren't valid on this type of swap chain. If you call any of these methods on this type of swap chain, they fail.

For info about how to choose a format for the swap chain's back buffer, see [Converting data for the color space](#).

Examples

For a code example showing how to use [CreateSwapChainForComposition](#), see [SwapChainPanel and gaming](#).

- For WinRT XAML, the [ISwapChainPanelNative](#) and [ISwapChainBackgroundPanelNative](#) interfaces are declared in the `windows.ui.xaml.media.dxinterop.h` header.
- For [Windows UI Library \(WinUI\)](#) XAML, the [ISwapChainPanelNative](#) and [ISwapChainBackgroundPanelNative](#) interfaces are declared in the `microsoft.ui.xaml.media.dxinterop.h` header.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

- For best performance, use DXGI flip model
 - IDXGIFactory2
-

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIFactory2::CreateSwapChainForCoreWindow method (dxgi1_2.h)

Article 10/13/2021

Creates a swap chain that is associated with the [CoreWindow](#) object for the output window for the swap chain.

Syntax

C++

```
HRESULT CreateSwapChainForCoreWindow(
    [in]          IUnknown                  *pDevice,
    [in]          IUnknown                  *pWindow,
    [in]          const DXGI_SWAP_CHAIN_DESC1 *pDesc,
    [in, optional] IDXGIOutput             *pRestrictToOutput,
    [out]         IDXGISwapChain1           **ppSwapChain
);
```

Parameters

[in] pDevice

For Direct3D 11, and earlier versions of Direct3D, this is a pointer to the Direct3D device for the swap chain. For Direct3D 12 this is a pointer to a direct command queue (refer to [ID3D12CommandQueue](#)). This parameter cannot be **NULL**.

[in] pWindow

A pointer to the [CoreWindow](#) object that is associated with the swap chain that [CreateSwapChainForCoreWindow](#) creates.

[in] pDesc

A pointer to a [DXGI_SWAP_CHAIN_DESC1](#) structure for the swap-chain description. This parameter cannot be **NULL**.

[in, optional] pRestrictToOutput

A pointer to the [IDXGIOutput](#) interface that the swap chain is restricted to. If the swap chain is moved to a different output, the content is black. You can optionally set this parameter to an output target that uses [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) to

restrict the content on this output. If you do not set this parameter to restrict content on an output target, you can set it to **NULL**.

[out] ppSwapChain

A pointer to a variable that receives a pointer to the [IDXGISwapChain1](#) interface for the swap chain that [CreateSwapChainForCoreWindow](#) creates.

Return value

[CreateSwapChainForCoreWindow](#) returns:

- **S_OK** if it successfully created a swap chain.
- **E_OUTOFMEMORY** if memory is unavailable to complete the operation.
- **DXGI_ERROR_INVALID_CALL** if the calling application provided invalid data, for example, if *pDesc* or *ppSwapChain* is **NULL**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic that are defined by the type of device that you pass to *pDevice*.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, [CreateSwapChainForCoreWindow](#) fails with **E_NOTIMPL**. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

Note Use this method in Windows Store apps rather than [IDXGIFactory2::CreateSwapChainForHwnd](#).

If you specify the width, height, or both (**Width** and **Height** members of [DXGI_SWAP_CHAIN_DESC1](#) that *pDesc* points to) of the swap chain as zero, the runtime obtains the size from the output window that the *pWindow* parameter specifies.

You can subsequently call the [IDXGISwapChain1::GetDesc1](#) method to retrieve the assigned width or height value.

Because you can associate only one flip presentation model swap chain (per layer) at a time with a [CoreWindow](#), the Microsoft Direct3D 11 policy of deferring the destruction of objects can cause problems if you attempt to destroy a flip presentation model swap chain and replace it with another swap chain. For more info about this situation, see [Deferred Destruction Issues with Flip Presentation Swap Chains](#).

For info about how to choose a format for the swap chain's back buffer, see [Converting data for the color space](#).

Overlapping swap chains

Starting with Windows 8.1, it is possible to create an additional swap chain in the foreground layer. A foreground swap chain can be used to render UI elements at native resolution while scaling up real-time rendering in the background swap chain (such as gameplay). This enables scenarios where lower resolution rendering is required for faster fill rates, but without sacrificing UI quality.

Foreground swap chains are created by setting the `DXGI_SWAP_CHAIN_FLAG_FOREGROUND_LAYER` swap chain flag in the `DXGI_SWAP_CHAIN_DESC1` that *pDesc* points to. Foreground swap chains must also use the `DXGI_ALPHA_MODE_PREMULTIPLIED` alpha mode, and must use `DXGI_SCALING_NONE`. Premultiplied alpha means that each pixel's color values are expected to be already multiplied by the alpha value before the frame is presented. For example, a 100% white BGRA pixel at 50% alpha is set to (0.5, 0.5, 0.5, 0.5). The alpha premultiplication step can be done in the output-merger stage by applying an app blend state (see [ID3D11BlendState](#)) with the `D3D11_RENDER_TARGET_BLEND_DESC` structure's `SrcBlend` field set to `D3D11_SRC_ALPHA`. If the alpha premultiplication step is not done, colors on the foreground swap chain will be brighter than expected.

The foreground swap chain will use multiplane overlays if supported by the hardware. Call [IDXGIOutput2::SupportsOverlays](#) to query the adapter for overlay support.

The following example creates a foreground swap chain for a CoreWindow:

C++

```
DXGI_SWAP_CHAIN_DESC1 swapChainDesc = { 0 };

swapChainDesc.Width = static_cast<UINT>(m_d3dRenderTargetSize.Width);
swapChainDesc.Height = static_cast<UINT>(m_d3dRenderTargetSize.Height);
swapChainDesc.Format = DXGI_FORMAT_B8G8R8A8_UNORM;
swapChainDesc.Stereo = false;
swapChainDesc.SampleDesc.Count = 1; // Don't use multi-sampling.
swapChainDesc.SampleDesc.Quality = 0;
swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
swapChainDesc.BufferCount = 2;
swapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL;
swapChainDesc.Flags = DXGI_SWAP_CHAIN_FLAG_FOREGROUND_LAYER;
swapChainDesc.AlphaMode = DXGI_ALPHA_MODE_PREMULTIPLIED;
swapChainDesc.Scaling = DXGI_SCALING_NONE;
```

```
ComPtr swapChain;
HRESULT hr = dxgiFactory->CreateSwapChainForCoreWindow(
    m_d3dDevice.Get(),
    reinterpret_cast<IUnknown*>(m_window.Get()),
    &swapChainDesc,
    nullptr,
    &swapChain
);
```

Present both swap chains together after rendering is complete.

The following example presents both swap chains:

C++

```
HRESULT hr = m_swapChain->Present(1, 0);

if (SUCCEEDED(hr) && m_foregroundSwapChain)
{
    m_foregroundSwapChain->Present(1, 0);
}
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[CoreWindow](#)

For best performance, use DXGI flip model

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::CreateSwapChainForHwnd method (dxgi1_2.h)

Article 10/13/2021

Creates a swap chain that is associated with an [HWND](#) handle to the output window for the swap chain.

Syntax

C++

```
HRESULT CreateSwapChainForHwnd(
    [in]          IUnknown* pDevice,
    [in]          HWND hWnd,
    [in]          const DXGI_SWAP_CHAIN_DESC1* pDesc,
    [in, optional] const DXGI_SWAP_CHAIN_FULLSCREEN_DESC* pFullscreenDesc,
    [in, optional] IDXGIOutput* pRestrictToOutput,
    [out]         IDXGISwapChain1** ppSwapChain
);
```

Parameters

[in] `pDevice`

For Direct3D 11, and earlier versions of Direct3D, this is a pointer to the Direct3D device for the swap chain. For Direct3D 12 this is a pointer to a direct command queue (refer to [ID3D12CommandQueue](#)). This parameter cannot be **NULL**.

[in] `hWnd`

The [HWND](#) handle that is associated with the swap chain that `CreateSwapChainForHwnd` creates. This parameter cannot be **NULL**.

[in] `pDesc`

A pointer to a [DXGI_SWAP_CHAIN_DESC1](#) structure for the swap-chain description. This parameter cannot be **NULL**.

[in, optional] `pFullscreenDesc`

A pointer to a [DXGI_SWAP_CHAIN_FULLSCREEN_DESC](#) structure for the description of a full-screen swap chain. You can optionally set this parameter to create a full-screen swap

chain. Set it to **NULL** to create a windowed swap chain.

[in, optional] *pRestrictToOutput*

A pointer to the [IDXGIOutput](#) interface for the output to restrict content to. You must also pass the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag in a [IDXGISwapChain1::Present1](#) call to force the content to appear blacked out on any other output. If you want to restrict the content to a different output, you must create a new swap chain. However, you can conditionally restrict content based on the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag.

Set this parameter to **NULL** if you don't want to restrict content to an output target.

[out] *ppSwapChain*

A pointer to a variable that receives a pointer to the [IDXGISwapChain1](#) interface for the swap chain that [CreateSwapChainForHwnd](#) creates.

Return value

[CreateSwapChainForHwnd](#) returns:

- **S_OK** if it successfully created a swap chain.
- **E_OUTOFMEMORY** if memory is unavailable to complete the operation.
- [DXGI_ERROR_INVALID_CALL](#) if the calling application provided invalid data, for example, if *pDesc* or *ppSwapChain* is **NULL**, or *pDesc* data members are invalid.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic that are defined by the type of device that you pass to *pDevice*.

Platform Update for Windows 7: [DXGI_SCALING_NONE](#) is not supported on Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed and causes [CreateSwapChainForHwnd](#) to return [DXGI_ERROR_INVALID_CALL](#) when called. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

Note Do not use this method in Windows Store apps. Instead, use [IDXGIFactory2::CreateSwapChainForCoreWindow](#).

If you specify the width, height, or both (**Width** and **Height** members of [DXGI_SWAP_CHAIN_DESC1](#) that *pDesc* points to) of the swap chain as zero, the runtime obtains the size from the output window that the *hWnd* parameter specifies.

You can subsequently call the [IDXGISwapChain1::GetDesc1](#) method to retrieve the assigned width or height value.

Because you can associate only one flip presentation model swap chain at a time with an [HWND](#), the Microsoft Direct3D 11 policy of deferring the destruction of objects can cause problems if you attempt to destroy a flip presentation model swap chain and replace it with another swap chain. For more info about this situation, see [Deferred Destruction Issues with Flip Presentation Swap Chains](#).

For info about how to choose a format for the swap chain's back buffer, see [Converting data for the color space](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[For best performance, use DXGI flip model](#)

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIFactory2::GetSharedResourceAdapterLuid method (dxgi1_2.h)

Article 10/13/2021

Identifies the adapter on which a shared resource object was created.

Syntax

C++

```
HRESULT GetSharedResourceAdapterLuid(
    [in]  HANDLE hResource,
    [out] LUID   *pLuid
);
```

Parameters

[in] *hResource*

A handle to a shared resource object. The [IDXGIResource1::CreateSharedHandle](#) method returns this handle.

[out] *pLuid*

A pointer to a variable that receives a locally unique identifier ([LUID](#)) value that identifies the adapter. **LUID** is defined in Dxgi.h. An **LUID** is a 64-bit value that is guaranteed to be unique only on the operating system on which it was generated. The uniqueness of an **LUID** is guaranteed only until the operating system is restarted.

Return value

GetSharedResourceAdapterLuid returns:

- S_OK if it identified the adapter.
- [DXGI_ERROR_INVALID_CALL](#) if *hResource* is invalid.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **GetSharedResourceAdapterLuid** fails with

E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

You cannot share resources across adapters. Therefore, you cannot open a shared resource on an adapter other than the adapter on which the resource was created. Call [GetSharedResourceAdapterLuid](#) before you open a shared resource to ensure that the resource was created on the appropriate adapter. To open a shared resource, call the [ID3D11Device1::OpenSharedResource1](#) or [ID3D11Device1::OpenSharedResourceByName](#) method.

Examples

syntax

```
HANDLE handle;
IDXGIFactory2* pFactory;

LUID luid;
pFactory->GetSharedResourceAdapterLuid (handle, &luid);

UINT index = 0;
IDXGIAdapter* pAdapter = NULL;
while (SUCCEEDED(pFactory->EnumAdapters(index, &pAdapter)))
{
    DXGI_ADAPTER_DESC desc;
    pAdapter->GetDesc(&desc);
    if (desc.AdapterLuid == luid)
    {
        // Identified a matching adapter.
        break;
    }
    pAdapter->Release();
    pAdapter = NULL;
    index++;
}
// At this point, if pAdapter is non-null, you identified an adapter that
// can open the shared resource.
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::IsWindowedStereoEnabled method (dxgi1_2.h)

Article 06/29/2021

Determines whether to use stereo mode.

Syntax

C++

```
BOOL IsWindowedStereoEnabled();
```

Return value

Indicates whether to use stereo mode. **TRUE** indicates that you can use stereo mode; otherwise, **FALSE**.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **IsWindowedStereoEnabled** always returns FALSE because stereoscopic 3D display behavior isn't available with the Platform Update for Windows 7. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

We recommend that windowed applications call **IsWindowedStereoEnabled** before they attempt to use stereo. **IsWindowedStereoEnabled** returns **TRUE** if both of the following items are true:

- All adapters in the computer have drivers that are capable of stereo. This only means that the driver is implemented to the Windows Display Driver Model (WDDM) for Windows 8 (WDDM 1.2). However, the adapter does not necessarily have to be able to scan out stereo.
- The current desktop mode (desktop modes are mono) and system policy and hardware are configured so that the Desktop Window Manager (DWM) performs stereo composition on at least one adapter output.

The creation of a windowed stereo swap chain succeeds if the first requirement is met. However, if the adapter can't scan out stereo, the output on that adapter is reduced to

mono.

The [Direct3D 11.1 Simple Stereo 3D Sample](#) shows how to add a stereoscopic 3D effect and how to respond to system stereo changes.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::RegisterOcclusionStatusEvent method (dxgi1_2.h)

Article 10/13/2021

Registers to receive notification of changes in occlusion status by using event signaling.

Syntax

C++

```
HRESULT RegisterOcclusionStatusEvent(
    [in]  HANDLE hEvent,
    [out] DWORD  *pdwCookie
);
```

Parameters

[in] *hEvent*

A handle to the event object that the operating system sets when notification of occlusion status change occurs. The [CreateEvent](#) or [OpenEvent](#) function returns this handle.

[out] *pdwCookie*

A pointer to a key value that an application can pass to the [IDXGIFactory2::UnregisterOcclusionStatus](#) method to unregister the notification event that *hEvent* specifies.

Return value

RegisterOcclusionStatusEvent returns:

- S_OK if the method successfully registered the event.
- E_OUTOFMEMORY if memory is unavailable to complete the operation.
- DXGI_ERROR_INVALID_CALL if *hEvent* is not a valid handle or not an event handle.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **RegisterOcclusionStatusEvent** fails with

E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

If you call `RegisterOcclusionStatusEvent` multiple times with the same event handle, `RegisterOcclusionStatusEvent` fails with `DXGI_ERROR_INVALID_CALL`.

If you call `RegisterOcclusionStatusEvent` multiple times with the different event handles, `RegisterOcclusionStatusEvent` properly registers the events.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::RegisterOcclusionStatusWindow method (dxgi1_2.h)

Article 10/13/2021

Registers an application window to receive notification messages of changes of occlusion status.

Syntax

C++

```
HRESULT RegisterOcclusionStatusWindow(
    [in]  HWND  WindowHandle,
    [in]  UINT  wMsg,
    [out] DWORD *pdwCookie
);
```

Parameters

[in] `WindowHandle`

The handle of the window to send a notification message to when occlusion status change occurs.

[in] `wMsg`

Identifies the notification message to send.

[out] `pdwCookie`

A pointer to a key value that an application can pass to the [IDXGIFactory2::UnregisterOcclusionStatus](#) method to unregister the notification message that `wMsg` specifies.

Return value

`RegisterOcclusionStatusWindow` returns:

- `S_OK` if it successfully registered the window.
- `E_OUTOFMEMORY` if memory is unavailable to complete the operation.

- [DXGI_ERROR_INVALID_CALL](#) if *WindowHandle* is not a valid window handle or not the window handle that the current process owns.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **RegisterOcclusionStatusWindow** fails with E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

Apps choose the Windows message that Windows sends when occlusion status changes.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::RegisterStereoStatusEvent method (dxgi1_2.h)

Article 10/13/2021

Registers to receive notification of changes in stereo status by using event signaling.

Syntax

C++

```
HRESULT RegisterStereoStatusEvent(
    [in]    HANDLE hEvent,
    [out]   DWORD  *pdwCookie
);
```

Parameters

[in] `hEvent`

A handle to the event object that the operating system sets when notification of stereo status change occurs. The [CreateEvent](#) or [OpenEvent](#) function returns this handle.

[out] `pdwCookie`

A pointer to a key value that an application can pass to the [IDXGIFactory2::UnregisterStereoStatus](#) method to unregister the notification event that *hEvent* specifies.

Return value

`RegisterStereoStatusEvent` returns:

- `S_OK` if it successfully registered the event.
- `E_OUTOFMEMORY` if memory is unavailable to complete the operation.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, `RegisterStereoStatusEvent` fails with `E_NOTIMPL`. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::RegisterStereoStatusWindow method (dxgi1_2.h)

Article 10/13/2021

Registers an application window to receive notification messages of changes of stereo status.

Syntax

C++

```
HRESULT RegisterStereoStatusWindow(
    [in]  HWND  WindowHandle,
    [in]  UINT  wMsg,
    [out] DWORD *pdwCookie
);
```

Parameters

[in] `WindowHandle`

The handle of the window to send a notification message to when stereo status change occurs.

[in] `wMsg`

Identifies the notification message to send.

[out] `pdwCookie`

A pointer to a key value that an application can pass to the [IDXGIFactory2::UnregisterStereoStatus](#) method to unregister the notification message that `wMsg` specifies.

Return value

`RegisterStereoStatusWindow` returns:

- `S_OK` if it successfully registered the window.
- `E_OUTOFMEMORY` if memory is unavailable to complete the operation.

- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, `RegisterStereoStatusWindow` fails with `E_NOTIMPL`. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	<code>dxgi1_2.h</code>
Library	<code>Dxgi.lib</code>

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::UnregisterOcclusionStatus method (dxgi1_2.h)

Article10/13/2021

Unregisters a window or an event to stop it from receiving notification when occlusion status changes.

Syntax

C++

```
void UnregisterOcclusionStatus(  
    [in] DWORD dwCookie  
);
```

Parameters

[in] dwCookie

A key value for the window or event to unregister. The [IDXGIFactory2::RegisterOcclusionStatusWindow](#) or [IDXGIFactory2::RegisterOcclusionStatusEvent](#) method returns this value.

Return value

None

Remarks

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **UnregisterOcclusionStatus** has no effect. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory2::UnregisterStereoStatus method (dxgi1_2.h)

Article 10/13/2021

Unregisters a window or an event to stop it from receiving notification when stereo status changes.

Syntax

C++

```
void UnregisterStereoStatus(  
    [in] DWORD dwCookie  
);
```

Parameters

[in] dwCookie

A key value for the window or event to unregister. The [IDXGIFactory2::RegisterStereoStatusWindow](#) or [IDXGIFactory2::RegisterStereoStatusEvent](#) method returns this value.

Return value

None

Remarks

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **UnregisterStereoStatus** has no effect. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput1 interface (dxgi1_2.h)

Article07/27/2022

An [IDXGIOutput1](#) interface represents an adapter output (such as a monitor).

Inheritance

The [IDXGIOutput1](#) interface inherits from [IDXGIOOutput](#). [IDXGIOutput1](#) also has these types of members:

Methods

The [IDXGIOutput1](#) interface has these methods.

IDXGIOutput1::DuplicateOutput
Creates a desktop duplication interface from the IDXGIOutput1 interface that represents an adapter output.
IDXGIOutput1::FindClosestMatchingMode1
Finds the display mode that most closely matches the requested display mode. (IDXGIOutput1.FindClosestMatchingMode1)
IDXGIOutput1::GetDisplayModeList1
Gets the display modes that match the requested format and other input options. (IDXGIOutput1.GetDisplayModeList1)
IDXGIOutput1::GetDisplaySurfaceData1
Copies the display surface (front buffer) to a user-provided resource.

Remarks

To determine the outputs that are available from the adapter, use [IDXGIAdapter::EnumOutputs](#). To determine the specific output that the swap chain will update, use [IDXGISwapChain::GetContainingOutput](#). You can then call [QueryInterface](#) from any [IDXGIOOutput](#) object to obtain an [IDXGIOutput1](#) object.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIOOutput](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput1::DuplicateOutput method (dxgi1_2.h)

Article 10/13/2021

Creates a desktop duplication interface from the [IDXGIOutput1](#) interface that represents an adapter output.

Syntax

C++

```
HRESULT DuplicateOutput(
    [in]     IUnknown                 *pDevice,
    [out]    IDXGIOutputDuplication **ppOutputDuplication
);
```

Parameters

[in] pDevice

A pointer to the Direct3D device interface that you can use to process the desktop image. This device must be created from the adapter to which the output is connected.

[out] ppOutputDuplication

A pointer to a variable that receives the new [IDXGIOutputDuplication](#) interface.

Return value

DuplicateOutput returns:

- S_OK if **DuplicateOutput** successfully created the desktop duplication interface.
- E_INVALIDARG for one of the following reasons:
 - The specified device (*pDevice*) is invalid, was not created on the correct adapter, or was not created from [IDXGIFactory1](#) (or a later version of a DXGI factory interface that inherits from [IDXGIFactory1](#)).
 - The calling application is already duplicating this desktop output.
- E_ACCESSDENIED if the application does not have access privilege to the current desktop image. For example, only an application that runs at LOCAL_SYSTEM can access the secure desktop.

- DXGI_ERROR_UNSUPPORTED if the created [IDXGIDuplicate](#) interface does not support the current desktop mode or scenario. For example, 8bpp and non-DWM desktop modes are not supported. If **DuplicateOutput** fails with DXGI_ERROR_UNSUPPORTED, the application can wait for system notification of desktop switches and mode changes and then call **DuplicateOutput** again after such a notification occurs. For more information, refer to [EVENT_SYSTEM_DESKTOPSWITCH](#) and mode change notification ([WM_DISPLAYCHANGE](#)).
- DXGI_ERROR_NOT_CURRENTLY_AVAILABLE if DXGI reached the limit on the maximum number of concurrent duplication applications (default of four). Therefore, the calling application cannot create any desktop duplication interfaces until the other applications close.
- DXGI_ERROR_SESSION_DISCONNECTED if **DuplicateOutput** failed because the session is currently disconnected.
- Other error codes are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **DuplicateOutput** fails with E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

If an application wants to duplicate the entire desktop, it must create a desktop duplication interface on each active output on the desktop. This interface does not provide an explicit way to synchronize the timing of each output image. Instead, the application must use the time stamp of each output, and then determine how to combine the images.

For **DuplicateOutput** to succeed, you must create *pDevice* from [IDXGIFactory1](#) or a later version of a DXGI factory interface that inherits from [IDXGIFactory1](#).

If the current mode is a stereo mode, the desktop duplication interface provides the image for the left stereo image only.

By default, only four processes can use a [IDXGIDuplicate](#) interface at the same time within a single session. A process can have only one desktop duplication interface on a single desktop output; however, that process can have a desktop duplication interface for each output that is part of the desktop.

For improved performance, consider using [DuplicateOutput1](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[DuplicateOutput1](#)

[IDXGIOutput1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput1::FindClosestMatchingMode1 method (dxgi1_2.h)

Article07/27/2022

Finds the display mode that most closely matches the requested display mode.

Syntax

C++

```
HRESULT FindClosestMatchingMode1(
    [in]           const DXGI_MODE_DESC1 *pModeToMatch,
    [out]          DXGI_MODE_DESC1      *pClosestMatch,
    [in, optional] IUnknown            *pConcernedDevice
);
```

Parameters

[in] pModeToMatch

A pointer to the [DXGI_MODE_DESC1](#) structure that describes the display mode to match. Members of [DXGI_MODE_DESC1](#) can be unspecified, which indicates no preference for that member. A value of 0 for **Width** or **Height** indicates that the value is unspecified. If either **Width** or **Height** is 0, both must be 0. A numerator and denominator of 0 in **RefreshRate** indicate it is unspecified. Other members of [DXGI_MODE_DESC1](#) have enumeration values that indicate that the member is unspecified. If *pConcernedDevice* is **NULL**, the **Format** member of [DXGI_MODE_DESC1](#) cannot be [DXGI_FORMAT_UNKNOWN](#).

[out] pClosestMatch

A pointer to the [DXGI_MODE_DESC1](#) structure that receives a description of the display mode that most closely matches the display mode described at *pModeToMatch*.

[in, optional] pConcernedDevice

A pointer to the Direct3D device interface. If this parameter is **NULL**, **FindClosestMatchingMode1** returns only modes whose format matches that of *pModeToMatch*; otherwise, **FindClosestMatchingMode1** returns only those formats that

are supported for scan-out by the device. For info about the formats that are supported for scan-out by the device at each feature level:

- DXGI Format Support for Direct3D Feature Level 12.1 Hardware
- DXGI Format Support for Direct3D Feature Level 12.0 Hardware
- DXGI Format Support for Direct3D Feature Level 11.1 Hardware
- DXGI Format Support for Direct3D Feature Level 11.0 Hardware
- Hardware Support for Direct3D 10Level9 Formats
- Hardware Support for Direct3D 10.1 Formats
- Hardware Support for Direct3D 10 Formats

Return value

Returns one of the error codes described in the [DXGI_ERROR](#) topic.

Remarks

Direct3D devices require UNORM formats.

FindClosestMatchingMode1 finds the closest matching available display mode to the mode that you specify in *pModeToMatch*.

If you set the **Stereo** member in the [DXGI_MODE_DESC1](#) structure to which *pModeToMatch* points to specify a stereo mode as input, **FindClosestMatchingMode1** considers only stereo modes. **FindClosestMatchingMode1** considers only mono modes if **Stereo** is not set.

FindClosestMatchingMode1 resolves similarly ranked members of display modes (that is, all specified, or all unspecified, and so on) in the following order:

1. **ScanlineOrdering**
2. **Scaling**
3. **Format**
4. **Resolution**
5. **RefreshRate**

When **FindClosestMatchingMode1** determines the closest value for a particular member, it uses previously matched members to filter the display mode list choices, and ignores other members. For example, when **FindClosestMatchingMode1** matches **Resolution**, it already filtered the display mode list by a certain **ScanlineOrdering**, **Scaling**, and **Format**, while it ignores **RefreshRate**. This ordering doesn't define the absolute ordering for every usage scenario of **FindClosestMatchingMode1**, because the

application can choose some values initially, which effectively changes the order of resolving members.

FindClosestMatchingMode1 matches members of the display mode one at a time, generally in a specified order.

If a member is unspecified, **FindClosestMatchingMode1** gravitates toward the values for the desktop related to this output. If this output is not part of the desktop, **FindClosestMatchingMode1** uses the default desktop output to find values. If an application uses a fully unspecified display mode, **FindClosestMatchingMode1** typically returns a display mode that matches the desktop settings for this output. Because unspecified members are lower priority than specified members, **FindClosestMatchingMode1** resolves unspecified members later than specified members.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	DXGI.lib

See also

[IDXGIOutput1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput1::GetDisplayModeList1 method (dxgi1_2.h)

Article 07/27/2022

Gets the display modes that match the requested format and other input options.

Syntax

C++

```
HRESULT GetDisplayModeList1(
    DXGI_FORMAT      EnumFormat,
    UINT             Flags,
    [in, out]        UINT          *pNumModes,
    [out, optional]  DXGI_MODE_DESC1 *pDesc
);
```

Parameters

EnumFormat

A [DXGI_FORMAT](#)-typed value for the color format.

Flags

A combination of [DXGI_ENUM_MODES](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for display modes to include. You must specify [DXGI_ENUM_MODES_SCALING](#) to expose the display modes that require scaling. Centered modes that require no scaling and correspond directly to the display output are enumerated by default.

[in, out] pNumModes

A pointer to a variable that receives the number of display modes that [GetDisplayModeList1](#) returns in the memory block to which *pDesc* points. Set *pDesc* to **NULL** so that *pNumModes* returns the number of display modes that match the format and the options. Otherwise, *pNumModes* returns the number of display modes returned in *pDesc*.

[out, optional] pDesc

A pointer to a list of display modes; set to **NULL** to get the number of display modes.

Return value

Returns one of the error codes described in the [DXGI_ERROR](#) topic. It is rare, but possible, that the display modes available can change immediately after calling this method, in which case **DXGI_ERROR_MORE_DATA** is returned (if there is not enough room for all the display modes).

Remarks

GetDisplayModeList1 is updated from [GetDisplayModeList](#) to return a list of [DXGI_MODE_DESC1](#) structures, which are updated mode descriptions. **GetDisplayModeList** behaves as though it calls **GetDisplayModeList1** because **GetDisplayModeList** can return all of the modes that are specified by [DXGI_ENUM_MODES](#), including stereo mode. However, **GetDisplayModeList** returns a list of [DXGI_MODE_DESC](#) structures, which are the former mode descriptions and do not indicate stereo mode.

The **GetDisplayModeList1** method does not enumerate stereo modes unless you specify the [DXGI_ENUM_MODES_STEREO](#) flag in the *Flags* parameter. If you specify [DXGI_ENUM_MODES_STEREO](#), stereo modes are included in the list of returned modes that the *pDesc* parameter points to. In other words, the method returns both stereo and mono modes.

In general, when you switch from windowed to full-screen mode, a swap chain automatically chooses a display mode that meets (or exceeds) the resolution, color depth, and refresh rate of the swap chain. To exercise more control over the display mode, use **GetDisplayModeList1** to poll the set of display modes that are validated against monitor capabilities, or all modes that match the desktop (if the desktop settings are not validated against the monitor).

The following example code shows that you need to call **GetDisplayModeList1** twice. First call **GetDisplayModeList1** to get the number of modes available, and second call **GetDisplayModeList1** to return a description of the modes.

```
UINT num = 0;
DXGI_FORMAT format = DXGI_FORMAT_R32G32B32A32_FLOAT;
UINT flags = DXGI_ENUM_MODES_INTERLACED;
```

```
p0Output->GetDisplayModeList1( format, flags, &num, 0);  
...  
DXGI_MODE_DESC1 * pDescs = new DXGI_MODE_DESC1[num];  
p0Output->GetDisplayModeList1( format, flags, &num, pDescs);
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	DXGI.lib

See also

[IDXGIOutput1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput1::GetDisplaySurfaceData1 method (dxgi1_2.h)

Article 10/13/2021

Copies the display surface (front buffer) to a user-provided resource.

Syntax

C++

```
HRESULT GetDisplaySurfaceData1(
    [in] IDXGIResource *pDestination
);
```

Parameters

[in] pDestination

A pointer to a resource interface that represents the resource to which `GetDisplaySurfaceData1` copies the display surface.

Return value

Returns one of the error codes described in the [DXGI_ERROR](#) topic.

Remarks

`GetDisplaySurfaceData1` is similar to [IDXGIOutput::GetDisplaySurfaceData](#) except `GetDisplaySurfaceData1` takes an [IDXGIResource](#) and `IDXGIOutput::GetDisplaySurfaceData` takes an [IDXGISurface](#).

`GetDisplaySurfaceData1` returns an error if the input resource is not a 2D texture (represented by the [ID3D11Texture2D](#) interface) with an array size (`ArraySize` member of the [D3D11_TEXTURE2D_DESC](#) structure) that is equal to the swap chain buffers.

The original [IDXGIOutput::GetDisplaySurfaceData](#) and the updated `GetDisplaySurfaceData1` behave exactly the same. `GetDisplaySurfaceData1` was required because textures with an array size equal to 2 (`ArraySize = 2`) do not implement [IDXGISurface](#).

You can call `GetDisplaySurfaceData1` only when an output is in full-screen mode. If `GetDisplaySurfaceData1` succeeds, it fills the destination resource.

Use `IDXGIOutput::GetDesc` to determine the size (width and height) of the output when you want to allocate space for the destination resource. This is true regardless of target monitor rotation. A destination resource created by a graphics component (such as Direct3D 11) must be created with CPU write permission (see `D3D11_CPU_ACCESS_WRITE`). Other surfaces can be created with CPU read-write permission (`D3D11_CPU_ACCESS_READ | D3D11_CPU_ACCESS_WRITE`).

`GetDisplaySurfaceData1` modifies the surface data to fit the destination resource (stretch, shrink, convert format, rotate). `GetDisplaySurfaceData1` performs the stretch and shrink with point sampling.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	<code>dxgi1_2.h</code>
Library	<code>DXGI.lib</code>

See also

[IDXGIOutput1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication interface (dxgi1_2.h)

Article 07/22/2021

The **IDXGIOutputDuplication** interface accesses and manipulates the duplicated desktop image.

Inheritance

The **IDXGIOutputDuplication** interface inherits from [IDXGIOBJECT](#).

IDXGIOutputDuplication also has these types of members:

Methods

The **IDXGIOutputDuplication** interface has these methods.

IDXGIOutputDuplication::AcquireNextFrame
Indicates that the application is ready to process the next desktop image.
IDXGIOutputDuplication::GetDesc
Retrieves a description of a duplicated output. This description specifies the dimensions of the surface that contains the desktop image.
IDXGIOutputDuplication::GetFrameDirtyRects
Gets information about dirty rectangles for the current desktop frame.
IDXGIOutputDuplication::GetFrameMoveRects
Gets information about the moved rectangles for the current desktop frame.
IDXGIOutputDuplication::GetFramePointerShape
Gets information about the new pointer shape for the current desktop frame.
IDXGIOutputDuplication::MapDesktopSurface
Provides the CPU with efficient access to a desktop image if that desktop image is already in system memory.

[IDXGIOutputDuplication::ReleaseFrame](#)

Indicates that the application finished processing the frame.

[IDXGIOutputDuplication::UnMapDesktopSurface](#)

Invalidates the pointer to the desktop image that was retrieved by using [IDXGIOutputDuplication::MapDesktopSurface](#).

Remarks

A collaboration application can use **IDXGIOutputDuplication** to access the desktop image. **IDXGIOutputDuplication** is supported in Desktop Window Manager (DWM) on non-8bpp DirectX full-screen modes and non-8bpp OpenGL full-screen modes. 16-bit or 32-bit GDI non-DWM desktop modes are not supported.

An application can use **IDXGIOutputDuplication** on a separate thread to receive the desktop images and to feed them into their specific image-processing pipeline. The application uses **IDXGIOutputDuplication** to perform the following operations:

1. Acquire the next desktop image.
2. Retrieve the information that describes the image.
3. Perform an operation on the image. This operation can be as simple as copying the image to a staging buffer so that the application can read the pixel data on the image. The application reads the pixel data after the application calls [IDXGISurface::Map](#). Alternatively, this operation can be more complex. For example, the application can run some pixel shaders on the updated regions of the image to encode those regions for transmission to a client.
4. After the application finishes processing each desktop image, it releases the image, loops to step 1, and repeats the steps. The application repeats these steps until it is finished processing desktop images.

The following components of the operating system can generate the desktop image:

- The DWM by composing the desktop image
- A full-screen DirectX or OpenGL application
- An application by switching to a separate desktop, for example, the secure desktop that is used to display the login screen

All current **IDXGIOutputDuplication** interfaces become invalid when the operating system switches to a different component that produces the desktop image or when a

mode change occurs. In these situations, the application must destroy its current **IDXGIOutputDuplication** interface and create a new **IDXGIOutputDuplication** interface.

Examples of situations in which **IDXGIOutputDuplication** becomes invalid are:

- Desktop switch
- Mode change
- Switch from DWM on, DWM off, or other full-screen application

In these situations, the application must release the **IDXGIOutputDuplication** interface and must create a new **IDXGIOutputDuplication** interface for the new content. If the application does not have the appropriate privilege to the new desktop image, its call to the [IDXGIOpen1::DuplicateOutput](#) method fails.

While the application processes each desktop image, the operating system accumulates all the desktop image updates into a single update. For more information about desktop updates, see [Updating the desktop image data](#).

The desktop image is always in the [DXGI_FORMAT_B8G8R8A8_UNORM](#) format.

The **IDXGIOutputDuplication** interface does not exist for Windows Store apps.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIOpen1](#)

Feedback



Was this page helpful? [!\[\]\(c75200dc97138db591df9f3f6645695f_img.jpg\) Yes](#) [!\[\]\(754bf5f4bdb03edf378e0bf9a972efc4_img.jpg\) No](#)

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::AcquireNextFrame method (dxgi1_2.h)

Article 10/13/2021

Indicates that the application is ready to process the next desktop image.

Syntax

C++

```
HRESULT AcquireNextFrame(
    [in]  UINT           TimeoutInMilliseconds,
    [out] DXGI_OUTDUPL_FRAME_INFO *pFrameInfo,
    [out] IDXGIResource   **ppDesktopResource
);
```

Parameters

[in] TimeoutInMilliseconds

The time-out interval, in milliseconds. This interval specifies the amount of time that this method waits for a new frame before it returns to the caller. This method returns if the interval elapses, and a new desktop image is not available.

For more information about the time-out interval, see Remarks.

[out] pFrameInfo

A pointer to a memory location that receives the [DXGI_OUTDUPL_FRAME_INFO](#) structure that describes timing and presentation statistics for a frame.

[out] ppDesktopResource

A pointer to a variable that receives the [IDXGIResource](#) interface of the surface that contains the desktop bitmap.

Return value

AcquireNextFrame returns:

- S_OK if it successfully received the next desktop image.

- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of image is displayed on the desktop. Examples of this situation are:
 - Desktop switch
 - Mode change
 - Switch from DWM on, DWM off, or other full-screen applicationIn this situation, the application must release the [IDXGIOutputDuplication](#) interface and create a new [IDXGIOutputDuplication](#) for the new content.
- DXGI_ERROR_WAIT_TIMEOUT if the time-out interval elapsed before the next desktop frame was available.
- DXGI_ERROR_INVALID_CALL if the application called [AcquireNextFrame](#) without releasing the previous frame.
- E_INVALIDARG if one of the parameters to [AcquireNextFrame](#) is incorrect; for example, if *pFrameInfo* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

When [AcquireNextFrame](#) returns successfully, the calling application can access the desktop image that [AcquireNextFrame](#) returns in the variable at *ppDesktopResource*. If the caller specifies a zero time-out interval in the *TimeoutInMilliseconds* parameter, [AcquireNextFrame](#) verifies whether there is a new desktop image available, returns immediately, and indicates its outcome with the return value. If the caller specifies an **INFINITE** time-out interval in the *TimeoutInMilliseconds* parameter, the time-out interval never elapses.

Note You cannot cancel the wait that you specified in the *TimeoutInMilliseconds* parameter. Therefore, if you must periodically check for other conditions (for example, a terminate signal), you should specify a non-**INFINITE** time-out interval. After the time-out interval elapses, you can check for these other conditions and then call [AcquireNextFrame](#) again to wait for the next frame.

[AcquireNextFrame](#) acquires a new desktop frame when the operating system either updates the desktop bitmap image or changes the shape or position of a hardware pointer. The new frame that [AcquireNextFrame](#) acquires might have only the desktop image updated, only the pointer shape or position updated, or both.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::GetDesc method (dxgi1_2.h)

Article 10/13/2021

Retrieves a description of a duplicated output. This description specifies the dimensions of the surface that contains the desktop image.

Syntax

C++

```
void GetDesc(  
    [out] DXGI_OUTDPL_DESC *pDesc  
);
```

Parameters

[out] pDesc

A pointer to a [DXGI_OUTDPL_DESC](#) structure that describes the duplicated output. This parameter must not be **NULL**.

Return value

None

Remarks

After an application creates an [IDXGIOutputDuplication](#) interface, it calls **GetDesc** to retrieve the dimensions of the surface that contains the desktop image. The format of the desktop image is always [DXGI_FORMAT_B8G8R8A8_UNORM](#).

Requirements

Minimum supported client

Windows 8 [desktop apps only]

Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::GetFrameDirtyRects method (dxgi1_2.h)

Article 10/13/2021

Gets information about dirty rectangles for the current desktop frame.

Syntax

C++

```
HRESULT GetFrameDirtyRects(
    [in]  UINT DirtyRectsBufferSize,
    [out] RECT *pDirtyRectsBuffer,
    [out] UINT *pDirtyRectsBufferSizeRequired
);
```

Parameters

[in] DirtyRectsBufferSize

The size in bytes of the buffer that the caller passed to the *pDirtyRectsBuffer* parameter.

[out] pDirtyRectsBuffer

A pointer to an array of [RECT](#) structures that identifies the dirty rectangle regions for the desktop frame.

[out] pDirtyRectsBufferSizeRequired

Pointer to a variable that receives the number of bytes that **GetFrameDirtyRects** needs to store information about dirty regions in the buffer at *pDirtyRectsBuffer*.

For more information about returning the required buffer size, see Remarks.

Return value

GetFrameDirtyRects returns:

- S_OK if it successfully retrieved information about dirty rectangles.
- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of

image is displayed on the desktop. Examples of this situation are:

- Desktop switch
- Mode change
- Switch from DWM on, DWM off, or other full-screen application

In this situation, the application must release the [IDXGIOutputDuplication](#) interface and create a new [IDXGIOutputDuplication](#) for the new content.

- DXGI_ERROR_MORE_DATA if the buffer that the calling application provided was not big enough.
- DXGI_ERROR_INVALID_CALL if the application called [GetFrameDirtyRects](#) without owning the desktop image.
- E_INVALIDARG if one of the parameters to [GetFrameDirtyRects](#) is incorrect; for example, if *pDirtyRectsBuffer* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

[GetFrameDirtyRects](#) stores a size value in the variable at *pDirtyRectsBufferSizeRequired*. This value specifies the number of bytes that [GetFrameDirtyRects](#) needs to store information about dirty regions. You can use this value in the following situations to determine the amount of memory to allocate for future buffers that you pass to *pDirtyRectsBuffer*:

- [GetFrameDirtyRects](#) fails with DXGI_ERROR_MORE_DATA because the buffer is not big enough.
- [GetFrameDirtyRects](#) supplies a buffer that is bigger than necessary. The size value returned at *pDirtyRectsBufferSizeRequired* informs the caller how much buffer space was actually used compared to how much buffer space the caller allocated and specified in the *DirtyRectsBufferSize* parameter.

The caller can also use the value returned at *pDirtyRectsBufferSizeRequired* to determine the number of [RECT](#)s returned in the *pDirtyRectsBuffer* array.

The buffer contains the list of dirty [RECT](#)s for the current frame.

Note To produce a visually accurate copy of the desktop, an application must first process all move [RECT](#)s before it processes dirty [RECT](#)s.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::GetFrameMoveRects method (dxgi1_2.h)

Article 10/13/2021

Gets information about the moved rectangles for the current desktop frame.

Syntax

C++

```
HRESULT GetFrameMoveRects(
    [in]  UINT           MoveRectsBufferSize,
    [out] DXGI_OUTDUPL_MOVE_RECT *pMoveRectBuffer,
    [out] UINT           *pMoveRectsBufferSizeRequired
);
```

Parameters

[in] MoveRectsBufferSize

The size in bytes of the buffer that the caller passed to the *pMoveRectBuffer* parameter.

[out] pMoveRectBuffer

A pointer to an array of [DXGI_OUTDUPL_MOVE_RECT](#) structures that identifies the moved rectangle regions for the desktop frame.

[out] pMoveRectsBufferSizeRequired

Pointer to a variable that receives the number of bytes that **GetFrameMoveRects** needs to store information about moved regions in the buffer at *pMoveRectBuffer*.

For more information about returning the required buffer size, see Remarks.

Return value

GetFrameMoveRects returns:

- S_OK if it successfully retrieved information about moved rectangles.
- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of

image is displayed on the desktop. Examples of this situation are:

- Desktop switch
- Mode change
- Switch from DWM on, DWM off, or other full-screen application

In this situation, the application must release the [IDXGIOOutputDuplication](#) interface and create a new [IDXGIOOutputDuplication](#) for the new content.

- DXGI_ERROR_MORE_DATA if the buffer that the calling application provided is not big enough.
- DXGI_ERROR_INVALID_CALL if the application called [GetFrameMoveRects](#) without owning the desktop image.
- E_INVALIDARG if one of the parameters to [GetFrameMoveRects](#) is incorrect; for example, if *pMoveRectBuffer* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

[GetFrameMoveRects](#) stores a size value in the variable at *pMoveRectsBufferSizeRequired*. This value specifies the number of bytes that [GetFrameMoveRects](#) needs to store information about moved regions. You can use this value in the following situations to determine the amount of memory to allocate for future buffers that you pass to *pMoveRectBuffer*:

- [GetFrameMoveRects](#) fails with DXGI_ERROR_MORE_DATA because the buffer is not big enough.
- [GetFrameMoveRects](#) supplies a buffer that is bigger than necessary. The size value returned at *pMoveRectsBufferSizeRequired* informs the caller how much buffer space was actually used compared to how much buffer space the caller allocated and specified in the *MoveRectsBufferSize* parameter.

The caller can also use the value returned at *pMoveRectsBufferSizeRequired* to determine the number of [DXGI_OUTDUPL_MOVE_RECT](#) structures returned.

The buffer contains the list of move RECTs for the current frame.

Note To produce a visually accurate copy of the desktop, an application must first process all move RECTs before it processes dirty RECTs.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::GetFramePointerShape method (dxgi1_2.h)

Article 10/13/2021

Gets information about the new pointer shape for the current desktop frame.

Syntax

C++

```
HRESULT GetFramePointerShape(
    [in]  UINT             PointerShapeBufferSize,
    [out] void*            *pPointerShapeBuffer,
    [out] UINT             *pPointerShapeBufferSizeRequired,
    [out] DXGI_OUTDUPL_POINTER_SHAPE_INFO *pPointerShapeInfo
);
```

Parameters

[in] PointerShapeBufferSize

The size in bytes of the buffer that the caller passed to the *pPointerShapeBuffer* parameter.

[out] pPointerShapeBuffer

A pointer to a buffer to which **GetFramePointerShape** copies and returns pixel data for the new pointer shape.

[out] pPointerShapeBufferSizeRequired

Pointer to a variable that receives the number of bytes that **GetFramePointerShape** needs to store the new pointer shape pixel data in the buffer at *pPointerShapeBuffer*.

For more information about returning the required buffer size, see Remarks.

[out] pPointerShapeInfo

Pointer to a [DXGI_OUTDUPL_POINTER_SHAPE_INFO](#) structure that receives the pointer shape information.

Return value

GetFramePointerShape returns:

- S_OK if it successfully retrieved information about the new pointer shape.
- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of image is displayed on the desktop. Examples of this situation are:
 - Desktop switch
 - Mode change
 - Switch from DWM on, DWM off, or other full-screen applicationIn this situation, the application must release the [IDXGIOutputDuplication](#) interface and create a new [IDXGIOutputDuplication](#) for the new content.
- DXGI_ERROR_MORE_DATA if the buffer that the calling application provided was not big enough.
- DXGI_ERROR_INVALID_CALL if the application called **GetFramePointerShape** without owning the desktop image.
- E_INVALIDARG if one of the parameters to **GetFramePointerShape** is incorrect; for example, if *pPointerShapeInfo* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

GetFramePointerShape stores a size value in the variable at *pPointerShapeBufferSizeRequired*. This value specifies the number of bytes that *pPointerShapeBufferSizeRequired* needs to store the new pointer shape pixel data. You can use the value in the following situations to determine the amount of memory to allocate for future buffers that you pass to *pPointerShapeBuffer*:

- **GetFramePointerShape** fails with DXGI_ERROR_MORE_DATA because the buffer is not big enough.
- **GetFramePointerShape** supplies a bigger than necessary buffer. The size value returned at *pPointerShapeBufferSizeRequired* informs the caller how much buffer space was actually used compared to how much buffer space the caller allocated and specified in the *PointerShapeBufferSize* parameter.

The *pPointerShapeInfo* parameter describes the new pointer shape.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::MapDesktopSurface method (dxgi1_2.h)

Article 10/13/2021

Provides the CPU with efficient access to a desktop image if that desktop image is already in system memory.

Syntax

C++

```
HRESULT MapDesktopSurface(
    [out] DXGI_MAPPED_RECT *pLockedRect
);
```

Parameters

[out] pLockedRect

A pointer to a [DXGI_MAPPED_RECT](#) structure that receives the surface data that the CPU needs to directly access the surface data.

Return value

MapDesktopSurface returns:

- S_OK if it successfully retrieved the surface data.
- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of image is displayed on the desktop. Examples of this situation are:
 - Desktop switch
 - Mode change
 - Switch from DWM on, DWM off, or other full-screen applicationIn this situation, the application must release the [IDXGIOutputDuplication](#) interface and create a new [IDXGIOutputDuplication](#) for the new content.
- DXGI_ERROR_INVALID_CALL if the application already has an outstanding map on the desktop image. The application must call [UnMapDesktopSurface](#) before it calls [MapDesktopSurface](#) again. DXGI_ERROR_INVALID_CALL is also returned if the application did not own the desktop image when it called [MapDesktopSurface](#).

- DXGI_ERROR_UNSUPPORTED if the desktop image is not in system memory. In this situation, the application must first transfer the image to a staging surface and then lock the image by calling the [IDXGISurface::Map](#) method.
- E_INVALIDARG if the *pLockedRect* parameter is incorrect; for example, if *pLockedRect* is **NULL**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

You can successfully call [MapDesktopSurface](#) if the **DesktopImageInSystemMemory** member of the [DXGI_OUTDUPL_DESC](#) structure is set to **TRUE**. If **DesktopImageInSystemMemory** is **FALSE**, [MapDesktopSurface](#) returns **DXGI_ERROR_UNSUPPORTED**. Call [IDXGIOutputDuplication::GetDesc](#) to retrieve the [DXGI_OUTDUPL_DESC](#) structure.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIOutputDuplication::ReleaseFrame method (dxgi1_2.h)

Article 06/29/2021

Indicates that the application finished processing the frame.

Syntax

C++

```
HRESULT ReleaseFrame();
```

Return value

ReleaseFrame returns:

- S_OK if it successfully completed.
- DXGI_ERROR_INVALID_CALL if the application already released the frame.
- DXGI_ERROR_ACCESS_LOST if the desktop duplication interface is invalid. The desktop duplication interface typically becomes invalid when a different type of image is displayed on the desktop. Examples of this situation are:
 - Desktop switch
 - Mode change
 - Switch from DWM on, DWM off, or other full-screen applicationIn this situation, the application must release the [IDXGIOutputDuplication](#) interface and create a new [IDXGIOutputDuplication](#) for the new content.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

The application must release the frame before it acquires the next frame. After the frame is released, the surface that contains the desktop bitmap becomes invalid; you will not be able to use the surface in a DirectX graphics operation.

For performance reasons, we recommend that you release the frame just before you call the [IDXGIOutputDuplication::AcquireNextFrame](#) method to acquire the next frame. When the client does not own the frame, the operating system copies all desktop updates to the surface. This can result in wasted GPU cycles if the operating system

updates the same region for each frame that occurs. When the client acquires the frame, the client is aware of only the final update to this region; therefore, any overlapping updates during previous frames are wasted. When the client acquires a frame, the client owns the surface; therefore, the operating system can track only the updated regions and cannot copy desktop updates to the surface. Because of this behavior, we recommend that you minimize the time between the call to release the current frame and the call to acquire the next frame.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutputDuplication::UnMapDesktopSurface method (dxgi1_2.h)

Article 06/29/2021

Invalidates the pointer to the desktop image that was retrieved by using [IDXGIOutputDuplication::MapDesktopSurface](#).

Syntax

C++

```
HRESULT UnMapDesktopSurface();
```

Return value

`UnMapDesktopSurface` returns:

- S_OK if it successfully completed.
- DXGI_ERROR_INVALID_CALL if the application did not map the desktop surface by calling [IDXGIOutputDuplication::MapDesktopSurface](#).
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIOutputDuplication](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource1 interface (dxgi1_2.h)

Article07/22/2021

An [IDXGIResource1](#) interface extends the [IDXGIResource](#) interface by adding support for creating a subresource surface object and for creating a handle to a shared resource.

Inheritance

The [IDXGIResource1](#) interface inherits from [IDXGIResource](#). [IDXGIResource1](#) also has these types of members:

Methods

The [IDXGIResource1](#) interface has these methods.

IDXGIResource1::CreateSharedHandle
Creates a handle to a shared resource. You can then use the returned handle with multiple Direct3D devices.
IDXGIResource1::CreateSubresourceSurface
Creates a subresource surface object.

Remarks

To determine the type of memory a resource is currently located in, use [IDXGIDevice::QueryResourceResidency](#). To share resources between processes, use [ID3D11Device1::OpenSharedResource1](#). For information about how to share resources between multiple Windows graphics APIs, including Direct3D 11, Direct2D, Direct3D 10, and Direct3D 9Ex, see [Surface Sharing Between Windows Graphics APIs](#).

You can retrieve the [IDXGIResource1](#) interface from any video memory resource that you create from a Direct3D 10 and later function. Any Direct3D object that supports [ID3D10Resource](#) or [ID3D11Resource](#) also supports [IDXGIResource1](#). For example, the Direct3D 2D texture object that you create from [ID3D11Device::CreateTexture2D](#) supports [IDXGIResource1](#). You can call [QueryInterface](#) on the 2D texture object ([ID3D11Texture2D](#)) to retrieve the [IDXGIResource1](#) interface. For example, to retrieve the [IDXGIResource1](#) interface from the 2D texture object, use the following code.

```
IDXGIResource1 * pDXGIResource;  
hr = g_pd3dTexture2D->QueryInterface(__uuidof(IDXGIResource1), (void**)&pDXGIResource);
```

Windows Phone 8: This API is supported.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource1::CreateSharedHandle method (dxgi1_2.h)

Article 05/24/2022

Creates a handle to a shared resource. You can then use the returned handle with multiple Direct3D devices.

Syntax

C++

```
HRESULT CreateSharedHandle(
    [in, optional] const SECURITY_ATTRIBUTES *pAttributes,
    [in]          DWORD                  dwAccess,
    [in, optional] LPCWSTR               lpName,
    [out]         HANDLE                *pHandle
);
```

Parameters

[in, optional] pAttributes

A pointer to a [SECURITY_ATTRIBUTES](#) structure that contains two separate but related data members: an optional security descriptor, and a Boolean value that determines whether child processes can inherit the returned handle.

Set this parameter to **NULL** if you want child processes that the application might create to not inherit the handle returned by [CreateSharedHandle](#), and if you want the resource that is associated with the returned handle to get a default security descriptor.

The **lpSecurityDescriptor** member of the structure specifies a [SECURITY_DESCRIPTOR](#) for the resource. Set this member to **NULL** if you want the runtime to assign a default security descriptor to the resource that is associated with the returned handle. The ACLs in the default security descriptor for the resource come from the primary or impersonation token of the creator. For more info, see [Synchronization Object Security and Access Rights](#).

[in] dwAccess

The requested access rights to the resource. In addition to the [generic access rights](#), DXGI defines the following values:

- **DXGI_SHARED_RESOURCE_READ** (0x80000000L) - specifies read access to the resource.
- **DXGI_SHARED_RESOURCE_WRITE** (1) - specifies write access to the resource.

You can combine these values by using a bitwise OR operation.

[in, optional] lpName

The name of the resource to share. The name is limited to MAX_PATH characters. Name comparison is case sensitive.

You will need the resource name if you call the [ID3D11Device1::OpenSharedResourceByName](#) method to access the shared resource by name. If you instead call the [ID3D11Device1::OpenSharedResource1](#) method to access the shared resource by handle, set this parameter to **NULL**.

If *lpName* matches the name of an existing resource, [CreateSharedHandle](#) fails with **DXGI_ERROR_NAME_ALREADY_EXISTS**. This occurs because these objects share the same namespace.

The name can have a "Global" or "Local" prefix to explicitly create the object in the global or session namespace. The remainder of the name can contain any character except the backslash character (\). For more information, see [Kernel Object Namespaces](#). Fast user switching is implemented using Terminal Services sessions. Kernel object names must follow the guidelines outlined for Terminal Services so that applications can support multiple users.

The object can be created in a private namespace. For more information, see [Object Namespaces](#).

[out] pHANDLE

A pointer to a variable that receives the NT HANDLE value to the resource to share. You can use this handle in calls to access the resource.

Return value

Returns S_OK if successful; otherwise, returns one of the following values:

- [DXGI_ERROR_INVALID_CALL](#) if one of the parameters is invalid.
- [DXGI_ERROR_NAME_ALREADY_EXISTS](#) if the supplied name of the resource to share is already associated with another resource.
- [E_ACCESSDENIED](#) if the object is being created in a protected namespace.
- [E_OUTOFMEMORY](#) if sufficient memory is not available to create the handle.

- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **CreateSharedHandle** fails with E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

CreateSharedHandle only returns the NT handle when you created the resource as shared and specified that it uses NT handles (that is, you set the [D3D11_RESOURCE_MISC_SHARED_NTHANDLE](#) and [D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX](#) flags). If you created the resource as shared and specified that it uses NT handles, you must use **CreateSharedHandle** to get a handle for sharing. In this situation, you can't use the [IDXGIResource::GetSharedHandle](#) method because it will fail.

You can pass the handle that **CreateSharedHandle** returns in a call to the [ID3D11Device1::OpenSharedResource1](#) method to give a device access to a shared resource that you created on a different device.

Because the handle that **CreateSharedHandle** returns is an NT handle, you can use the handle with [CloseHandle](#), [DuplicateHandle](#), and so on. You can call **CreateSharedHandle** only once for a shared resource; later calls fail. If you need more handles to the same shared resource, call [DuplicateHandle](#). When you no longer need the shared resource handle, call [CloseHandle](#) to close the handle, in order to avoid memory leaks.

If you pass a name for the resource to *lpName* when you call **CreateSharedHandle** to share the resource, you can subsequently pass this name in a call to the [ID3D11Device1::OpenSharedResourceByName](#) method to give another device access to the shared resource. If you use a named resource, a malicious user can use this named resource before you do and prevent your app from starting. To prevent this situation, create a randomly named resource and store the name so that it can only be obtained by an authorized user. Alternatively, you can use a file for this purpose. To limit your app to one instance per user, create a locked file in the user's profile directory.

If you created the resource as shared and did not specify that it uses NT handles, you cannot use **CreateSharedHandle** to get a handle for sharing because **CreateSharedHandle** will fail.

Examples

syntax

```
ID3D11Texture2D* pTexture2D;  
ID3D11Device* pDevice;  
  
pDevice->CreateTexture2D(..., &pTexture2D); // Create the texture as shared  
with NT HANDLES.  
  
HANDLE handle;  
IDXGIResource1* pResource;  
pTexture2D->QueryInterface(__uuidof(IDXGIResource1), (void**) &pResource);  
pResource->CreateSharedHandle(NULL,  
    DXGI_SHARED_RESOURCE_READ | DXGI_SHARED_RESOURCE_WRITE,  
    NULL,  
    &handle);  
  
// Pass the handle to another process to share the resource.
```

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIResource1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIResource1::CreateSubresourceSurface method (dxgi1_2.h)

Article 10/13/2021

Creates a subresource surface object.

Syntax

C++

```
HRESULT CreateSubresourceSurface(
    UINT           index,
    [out] IDXGISurface2 **ppSurface
);
```

Parameters

`index`

The index of the subresource surface object to enumerate.

`[out] ppSurface`

The address of a pointer to a [IDXGISurface2](#) interface that represents the created subresource surface object at the position specified by the *index* parameter.

Return value

Returns S_OK if successful; otherwise, returns one of the following values:

- [DXGI_ERROR_INVALID_CALL](#) if the index is out of range or if the subresource is not a valid surface.
- E_OUTOFMEMORY if insufficient memory is available to create the subresource surface object.

A subresource is a valid surface if the original resource would have been a valid surface had its array size been equal to 1.

Remarks

Subresource surface objects implement the [IDXGISurface2](#) interface, which inherits from [IDXGISurface1](#) and indirectly [IDXGISurface](#). Therefore, the GDI-interoperable methods of [IDXGISurface1](#) work if the original resource interface object was created with the GDI-interoperable flag ([D3D11_RESOURCE_MISC_GDI_COMPATIBLE](#)).

[CreateSubresourceSurface](#) creates a subresource surface that is based on the resource interface on which [CreateSubresourceSurface](#) is called. For example, if the original resource interface object is a 2D texture, the created subresource surface is also a 2D texture.

You can use [CreateSubresourceSurface](#) to create parts of a stereo resource so you can use Direct2D on either the left or right part of the stereo resource.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGIResource1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface2 interface (dxgi1_2.h)

Article07/22/2021

The [IDXGISurface2](#) interface extends the [IDXGISurface1](#) interface by adding support for subresource surfaces and getting a handle to a shared resource.

Inheritance

The [IDXGISurface2](#) interface inherits from [IDXGISurface1](#). [IDXGISurface2](#) also has these types of members:

Methods

The [IDXGISurface2](#) interface has these methods.

[IDXGISurface2::GetResource](#)

Gets the parent resource and subresource index that support a subresource surface.

Remarks

An image-data object is a 2D section of memory, commonly called a surface. To get the surface from an output, call [IDXGIOutput::GetDisplaySurfaceData](#). Then, call [QueryInterface](#) on the [IDXGISurface](#) object that [IDXGIOutput::GetDisplaySurfaceData](#) returns to retrieve the [IDXGISurface2](#) interface.

Any object that supports [IDXGISurface](#) also supports [IDXGISurface2](#).

The runtime automatically creates an [IDXGISurface2](#) interface when it creates a Direct3D resource object that represents a surface. For example, the runtime creates an [IDXGISurface2](#) interface when you call [ID3D11Device::CreateTexture2D](#) to create a 2D texture. To retrieve the [IDXGISurface2](#) interface that represents the 2D texture surface, call [ID3D11Texture2D::QueryInterface](#). In this call, you must pass the identifier of [IDXGISurface2](#). If the 2D texture has only a single MIP-map level and does not consist of an array of textures, [QueryInterface](#) succeeds and returns a pointer to the [IDXGISurface2](#) interface pointer. Otherwise, [QueryInterface](#) fails and does not return the pointer to [IDXGISurface2](#).

You can call the [IDXGIResource1::CreateSubresourceSurface](#) method to create an [IDXGISurface2](#) interface that refers to one subresource of a stereo resource.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGISurface1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISurface2::GetResource method (dxgi1_2.h)

Article 10/13/2021

Gets the parent resource and subresource index that support a subresource surface.

Syntax

C++

```
HRESULT GetResource(
    [in] REFIID riid,
    [out] void    **ppParentResource,
    [out] UINT     *pSubresourceIndex
);
```

Parameters

[in] `riid`

The globally unique identifier (GUID) of the requested interface type.

[out] `ppParentResource`

A pointer to a buffer that receives a pointer to the parent resource object for the subresource surface.

[out] `pSubresourceIndex`

A pointer to a variable that receives the index of the subresource surface.

Return value

Returns S_OK if successful; otherwise, returns one of the following values:

- E_NOINTERFACE if the object does not implement the GUID that the *riid* parameter specifies.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

For subresource surface objects that the [IDXGIResource1::CreateSubresourceSurface](#) method creates, **GetResource** simply returns the values that were used to create the subresource surface.

Current objects that implement [IDXGISurface](#) are either resources or views. **GetResource** for these objects returns “this” or the resource that supports the view respectively. In this situation, the subresource index is 0.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISurface2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1 interface (dxgi1_2.h)

Article07/22/2021

Provides presentation capabilities that are enhanced from [IDXGISwapChain](#). These presentation capabilities consist of specifying dirty rectangles and scroll rectangle to optimize the presentation.

Inheritance

The [IDXGISwapChain1](#) interface inherits from [IDXGISwapChain](#). [IDXGISwapChain1](#) also has these types of members:

Methods

The [IDXGISwapChain1](#) interface has these methods.

IDXGISwapChain1::GetBackgroundColor
Retrieves the background color of the swap chain.
IDXGISwapChain1::GetCoreWindow
Retrieves the underlying CoreWindow object for this swap-chain object.
IDXGISwapChain1::GetDesc1
Gets a description of the swap chain.
IDXGISwapChain1::GetFullscreenDesc
Gets a description of a full-screen swap chain.
IDXGISwapChain1::GetHwnd
Retrieves the underlying HWND for this swap-chain object.
IDXGISwapChain1::GetRestrictToOutput
Gets the output (the display monitor) to which you can restrict the contents of a present operation.

[IDXGISwapChain1::GetRotation](#)

Gets the rotation of the back buffers for the swap chain.

[IDXGISwapChain1::IsTemporaryMonoSupported](#)

Determines whether a swap chain supports "temporary mono."

[IDXGISwapChain1::Present1](#)

Presents a frame on the display screen.

[IDXGISwapChain1::SetBackgroundColor](#)

Changes the background color of the swap chain.

[IDXGISwapChain1::SetRotation](#)

Sets the rotation of the back buffers for the swap chain.

Remarks

You can create a swap chain by calling [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#). You can also create a swap chain when you call [D3D11CreateDeviceAndSwapChain](#); however, you can then only access the subset of swap-chain functionality that the [IDXGISwapChain](#) interface provides.

IDXGISwapChain1 provides the [IsTemporaryMonoSupported](#) method that you can use to determine whether the swap chain supports "temporary mono" presentation. This type of swap chain is a stereo swap chain that can be used to present mono content.

Note Some stereo features like the advanced presentation flags are not represented by an explicit interface change. Furthermore, the original ([IDXGISwapChain](#)) and new ([IDXGISwapChain1](#)) swap chain interfaces generally have the same behavior. For information about how [IDXGISwapChain](#) methods are translated into [IDXGISwapChain1](#) methods, see the descriptions of the [IDXGISwapChain1](#) methods.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

See also

[DXGI Interfaces](#)

[IDXGIFactory2::CreateSwapChainForComposition](#)

[IDXGIFactory2::CreateSwapChainForCoreWindow](#)

[IDXGIFactory2::CreateSwapChainForHwnd](#)

[IDXGISwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetBackgroundColor method (dxgi1_2.h)

Article 10/13/2021

Retrieves the background color of the swap chain.

Syntax

C++

```
HRESULT GetBackgroundColor(  
    [out] DXGI_RGBA *pColor  
) ;
```

Parameters

[out] *pColor*

A pointer to a [DXGI_RGBA](#) structure that receives the background color of the swap chain.

Return value

GetBackgroundColor returns:

- S_OK if it successfully retrieves the background color.
- [DXGI_ERROR_INVALID_CALL](#) if the *pColor* parameter is invalid, for example, *pColor* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

Note The background color that **GetBackgroundColor** retrieves does not indicate what the screen currently displays. The background color indicates what the screen will display with your next call to the **IDXGISwapChain1::Present1** method. The default value of the background color is black with full opacity: 0,0,0,1.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

[IDXGISwapChain1::SetBackgroundColor](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetCoreWindow method (dxgi1_2.h)

Article 10/13/2021

Retrieves the underlying [CoreWindow](#) object for this swap-chain object.

Syntax

C++

```
HRESULT GetCoreWindow(
    [in] REFIID refiid,
    [out] void    **ppUnk
);
```

Parameters

[in] `refiid`

A pointer to the globally unique identifier (GUID) of the [CoreWindow](#) object that is referenced by the *ppUnk* parameter.

[out] `ppUnk`

A pointer to a variable that receives a pointer to the [CoreWindow](#) object.

Return value

`GetCoreWindow` returns:

- `S_OK` if it successfully retrieved the underlying [CoreWindow](#) object.
- `DXGI_ERROR_INVALID_CALL` if *ppUnk* is `NULL`; that is, the swap chain is not associated with a [CoreWindow](#) object.
- Any [HRESULT](#) that a call to [QueryInterface](#) to query for an [CoreWindow](#) object might typically return.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, `GetCoreWindow` fails with `E_NOTIMPL`. For

more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

Applications call the [IDXGIFactory2::CreateSwapChainForCoreWindow](#) method to create a swap chain that is associated with an [CoreWindow](#) object.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetDesc1 method (dxgi1_2.h)

Article 10/13/2021

Gets a description of the swap chain.

Syntax

C++

```
HRESULT GetDesc1(  
    [out] DXGI_SWAP_CHAIN_DESC1 *pDesc  
>;
```

Parameters

[out] pDesc

A pointer to a [DXGI_SWAP_CHAIN_DESC1](#) structure that describes the swap chain.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetFullscreenDesc method (dxgi1_2.h)

Article 10/13/2021

Gets a description of a full-screen swap chain.

Syntax

C++

```
HRESULT GetFullscreenDesc(  
    [out] DXGI_SWAP_CHAIN_FULLSCREEN_DESC *pDesc  
) ;
```

Parameters

[out] pDesc

A pointer to a [DXGI_SWAP_CHAIN_FULLSCREEN_DESC](#) structure that describes the full-screen swap chain.

Return value

GetFullscreenDesc returns:

- S_OK if it successfully retrieved the description of the full-screen swap chain.
- [DXGI_ERROR_INVALID_CALL](#) for non-HWND swap chains or if *pDesc* is NULL.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Remarks

The semantics of **GetFullscreenDesc** are identical to that of the [IDXGISwapchain::GetDesc](#) method for [HWND](#)-based swap chains.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetHwnd method (dxgi1_2.h)

Article 10/13/2021

Retrieves the underlying [HWND](#) for this swap-chain object.

Syntax

C++

```
HRESULT GetHwnd(
    [out] HWND *pHwnd
);
```

Parameters

[out] pHwnd

A pointer to a variable that receives the [HWND](#) for the swap-chain object.

Return value

Returns [S_OK](#) if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

If *pHwnd* receives **NULL** (that is, the swap chain is not [HWND](#)-based), **GetHwnd** returns [DXGI_ERROR_INVALID_CALL](#).

Remarks

Applications call the [IDXGIFactory2::CreateSwapChainForHwnd](#) method to create a swap chain that is associated with an [HWND](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetRestrictToOutput method (dxgi1_2.h)

Article 10/13/2021

Gets the output (the display monitor) to which you can restrict the contents of a present operation.

Syntax

C++

```
HRESULT GetRestrictToOutput(  
    [out] IDXGIOutput **ppRestrictToOutput  
) ;
```

Parameters

[out] ppRestrictToOutput

A pointer to a buffer that receives a pointer to the [IDXGIOOutput](#) interface for the restrict-to output. An application passes this pointer to [IDXGIOOutput](#) in a call to the [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#) method to create the swap chain.

Return value

Returns S_OK if the restrict-to output was successfully retrieved; otherwise, returns E_INVALIDARG if the pointer is invalid.

Remarks

If the method succeeds, the runtime fills the buffer at *ppRestrictToOutput* with a pointer to the restrict-to output interface. This restrict-to output interface has its reference count incremented. When you are finished with it, be sure to release the interface to avoid a memory leak.

The output is also owned by the adapter on which the swap chain's device was created.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::GetRotation method (dxgi1_2.h)

Article 10/13/2021

Gets the rotation of the back buffers for the swap chain.

Syntax

C++

```
HRESULT GetRotation(  
    [out] DXGI_MODE_ROTATION *pRotation  
);
```

Parameters

[out] pRotation

A pointer to a variable that receives a [DXGI_MODE_ROTATION](#)-typed value that specifies the rotation of the back buffers for the swap chain.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **GetRotation** fails with DXGI_ERROR_INVALID_CALL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
--------------------------	---

Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::IsTemporaryMonoSupported method (dxgi1_2.h)

Article06/29/2021

Determines whether a swap chain supports “temporary mono.”

Syntax

C++

```
BOOL IsTemporaryMonoSupported();
```

Return value

Indicates whether to use the swap chain in temporary mono mode. **TRUE** indicates that you can use temporary-mono mode; otherwise, **FALSE**.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **IsTemporaryMonoSupported** always returns FALSE because stereoscopic 3D display behavior isn’t available with the Platform Update for Windows 7. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

Temporary mono is a feature where a stereo swap chain can be presented using only the content in the left buffer. To present using the left buffer as a mono buffer, an application calls the [IDXGISwapChain1::Present1](#) method with the [DXGI_PRESENT_STEREO_TEMPORARY_MONO](#) flag. All windowed swap chains support temporary mono. However, full-screen swap chains optionally support temporary mono because not all hardware supports temporary mono on full-screen swap chains efficiently.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::Present1 method (dxgi1_2.h)

Article 10/13/2021

Presents a frame on the display screen.

Syntax

C++

```
HRESULT Present1(
    UINT                     SyncInterval,
    UINT                     PresentFlags,
    [in] const DXGI_PRESENT_PARAMETERS *pPresentParameters
);
```

Parameters

SyncInterval

An integer that specifies how to synchronize presentation of a frame with the vertical blank.

For the bit-block transfer (bitblt) model ([DXGI_SWAP_EFFECT_DISCARD](#) or [DXGI_SWAP_EFFECT_SEQUENTIAL](#)), values are:

- 0 - The presentation occurs immediately, there is no synchronization.
- 1 through 4 - Synchronize presentation after the *n*th vertical blank.

For the flip model ([DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#)), values are:

- 0 - Cancel the remaining time on the previously presented frame and discard this frame if a newer frame is queued.
- 1 through 4 - Synchronize presentation for at least *n* vertical blanks.

For an example that shows how sync-interval values affect a flip presentation queue, see Remarks.

If the update region straddles more than one output (each represented by [IDXGIOutput1](#)), **Present1** performs the synchronization to the output that contains the largest sub-rectangle of the target window's client area.

PresentFlags

An integer value that contains swap-chain presentation options. These options are defined by the [DXGI_PRESENT](#) constants.

[in] pPresentParameters

A pointer to a [DXGI_PRESENT_PARAMETERS](#) structure that describes updated rectangles and scroll information of the frame to present.

Return value

Possible return values include: S_OK, [DXGI_ERROR_DEVICE_REMOVED](#), [DXGI_STATUS_OCCLUDED](#), [DXGI_ERROR_INVALID_CALL](#), or E_OUTOFMEMORY.

Remarks

An app can use [Present1](#) to optimize presentation by specifying scroll and dirty rectangles. When the runtime has information about these rectangles, the runtime can then perform necessary bitblts during presentation more efficiently and pass this metadata to the Desktop Window Manager (DWM). The DWM can then use the metadata to optimize presentation and pass the metadata to indirect displays and terminal servers to optimize traffic over the wire. An app must confine its modifications to only the dirty regions that it passes to [Present1](#), as well as modify the entire dirty region to avoid undefined resource contents from being exposed.

For flip presentation model swap chains that you create with the [DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#) value set, a successful presentation results in an unbind of back buffer 0 from the graphics pipeline, except for when you pass the [DXGI_PRESENT_DO_NOT_SEQUENCE](#) flag in the *Flags* parameter.

For info about how data values change when you present content to the screen, see [Converting data for the color space](#).

For info about calling [Present1](#) when your app uses multiple threads, see [Multithread Considerations](#) and [Multithreading and DXGI](#).

Flip presentation model queue

Suppose the following frames with sync-interval values are queued from oldest (A) to newest (E) before you call [Present1](#).

A: 3, B: 0, C: 0, D: 1, E: 0

When you call **Present1**, the runtime shows frame A for only 1 vertical blank interval. The runtime terminates frame A early because of the sync interval 0 in frame B. Then the runtime shows frame D for 1 vertical blank interval, and then frame E until you submit a new presentation. The runtime discards frames B and C.

Variable refresh rate displays

It is a requirement of variable refresh rate displays that tearing is enabled. The [CheckFeatureSupport](#) method can be used to determine if this feature is available, and to set the required flags refer to the descriptions of [DXGI_PRESENT_ALLOW_TEARING](#) and [DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING](#), and the **Variable refresh rate displays/Vsync off** section of [DXGI 1.5 Improvements](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

[Present](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::SetBackgroundColor method (dxgi1_2.h)

Article 10/13/2021

Changes the background color of the swap chain.

Syntax

C++

```
HRESULT SetBackgroundColor(  
    [in] const DXGI_RGBA *pColor  
);
```

Parameters

[in] pColor

A pointer to a [DXGI_RGBA](#) structure that specifies the background color to set.

Return value

`SetBackgroundColor` returns:

- S_OK if it successfully set the background color.
- E_INVALIDARG if the *pColor* parameter is incorrect, for example, *pColor* is NULL or any of the floating-point values of the members of [DXGI_RGBA](#) to which *pColor* points are outside the range from 0.0 through 1.0.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, `SetBackgroundColor` fails with E_NOTIMPL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

The background color affects only swap chains that you create with [DXGI_SCALING_NONE](#) in windowed mode. You pass this value in a call to

[IDXGIFactory2::CreateSwapChainForHwnd](#),
[IDXGIFactory2::CreateSwapChainForCoreWindow](#), or
[IDXGIFactory2::CreateSwapChainForComposition](#). Typically, the background color is not visible unless the swap-chain contents are smaller than the destination window.

When you set the background color, it is not immediately realized. It takes effect in conjunction with your next call to the [IDXGISwapChain1::Present1](#) method. The [DXGI_PRESENT](#) flags that you pass to [IDXGISwapChain1::Present1](#) can help achieve the effect that you require. For example, if you call [SetBackgroundColor](#) and then call [IDXGISwapChain1::Present1](#) with the *Flags* parameter set to [DXGI_PRESENT_DO_NOT_SEQUENCE](#), you change only the background color without changing the displayed contents of the swap chain.

When you call the [IDXGISwapChain1::Present1](#) method to display contents of the swap chain, [IDXGISwapChain1::Present1](#) uses the [DXGI_ALPHA_MODE](#) value that is specified in the [AlphaMode](#) member of the [DXGI_SWAP_CHAIN_DESC1](#) structure to determine how to handle the *a* member of the [DXGI_RGBA](#) structure, the alpha value of the background color, that achieves window transparency. For example, if [AlphaMode](#) is [DXGI_ALPHA_MODE_IGNORE](#), [IDXGISwapChain1::Present1](#) ignores the *a* member of [DXGI_RGBA](#).

Note Like all presentation data, we recommend that you perform floating point operations in a linear color space. When the desktop is in a fixed bit color depth mode, the operating system converts linear color data to standard RGB data (sRGB, gamma 2.2 corrected space) to compose to the screen. For more info, see [Converting data for the color space](#).

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h

Library

Dxgi.lib

See also

[DXGI_SCALING](#)

[IDXGISwapChain1](#)

[IDXGISwapChain1::GetBackgroundColor](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain1::SetRotation method (dxgi1_2.h)

Article 10/13/2021

Sets the rotation of the back buffers for the swap chain.

Syntax

C++

```
HRESULT SetRotation(  
    [in] DXGI_MODE_ROTATION Rotation  
);
```

Parameters

[in] Rotation

A [DXGI_MODE_ROTATION](#)-typed value that specifies how to set the rotation of the back buffers for the swap chain.

Return value

SetRotation returns:

- S_OK if it successfully set the rotation.
- DXGI_ERROR_INVALID_CALL if the swap chain is bit-block transfer (bitblt) model.
The swap chain must be flip model to successfully call **SetRotation**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Platform Update for Windows 7: On Windows 7 or Windows Server 2008 R2 with the [Platform Update for Windows 7](#) installed, **SetRotation** fails with DXGI_ERROR_INVALID_CALL. For more info about the Platform Update for Windows 7, see [Platform Update for Windows 7](#).

Remarks

You can only use **SetRotation** to rotate the back buffers for flip-model swap chains that you present in windowed mode.

SetRotation isn't supported for rotating the back buffers for flip-model swap chains that you present in full-screen mode. In this situation, **SetRotation** doesn't fail, but you must ensure that you specify no rotation ([DXGI_MODE_ROTATION_IDENTITY](#)) for the swap chain. Otherwise, when you call [IDXGISwapChain1::Present1](#) or [IDXGISwapChain::Present](#) to present a frame, the presentation fails.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_2.h
Library	Dxgi.lib

See also

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgi1_3.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi1_3.h contains the following programming interfaces:

Interfaces

[IDXGIDecodeSwapChain](#)

Represents a swap chain that is used by desktop media apps to decode video data and show it on a DirectComposition surface.

[IDXGIDevice3](#)

The IDXGIDevice3 interface implements a derived class for DXGI objects that produce image data. The interface exposes a method to trim graphics memory usage by the DXGI device.

[IDXGIFactory3](#)

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects. (IDXGIFactory3)

[IDXGIFactoryMedia](#)

Creates swap chains for desktop media apps that use DirectComposition surfaces to decode and display video.

[IDXGIOutput2](#)

Represents an adapter output (such as a monitor). The IDXGIOutput2 interface exposes a method to check for multiplane overlay support on the primary output adapter.

[IDXGIOutput3](#)

Represents an adapter output (such as a monitor). The IDXGIOutput3 interface exposes a method to check for overlay support.

[IDXGISwapChain2](#)

Extends IDXGISwapChain1 with methods to support swap back buffer scaling and lower-latency swap chains.

[IDXGISwapChainMedia](#)

This swap chain interface allows desktop media applications to request a seamless change to a specific refresh rate.

Functions

[CreateDXGIFactory2](#)

Creates a DXGI 1.3 factory that you can use to generate other DXGI objects.

[DXGIGetDebugInterface1](#)

Retrieves an interface that Windows Store apps use for debugging the Microsoft DirectX Graphics Infrastructure (DXGI).

Structures

[DXGI_DECODE_SWAP_CHAIN_DESC](#)

Used with IDXGIFactoryMedia::CreateDecodeSwapChainForCompositionSurfaceHandle to describe a decode swap chain.

[DXGI_FRAME_STATISTICS_MEDIA](#)

Used to verify system approval for the app's custom present duration (custom refresh rate).

[DXGI_MATRIX_3X2_F](#)

Represents a 3x2 matrix. Used with GetMatrixTransform and SetMatrixTransform to indicate the scaling and translation transform for SwapChainPanel swap chains.

Enumerations

[DXGI_FRAME_PRESENTATION_MODE](#)

Indicates options for presenting frames to the swap chain.

[DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS](#)

Options for swap-chain color space.

[DXGI_OVERLAY_SUPPORT_FLAG](#)

Specifies overlay support to check for in a call to IDXGIOOutput3::CheckOverlaySupport.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

CreateDXGIFactory2 function (dxgi1_3.h)

Article10/13/2021

Creates a DXGI 1.3 factory that you can use to generate other DXGI objects.

In Windows 8, any DXGI factory created while DXGIDebug.dll was present on the system would load and use it. Starting in Windows 8.1, apps explicitly request that DXGIDebug.dll be loaded instead. Use **CreateDXGIFactory2** and specify the DXGI_CREATE_FACTORY_DEBUG flag to request DXGIDebug.dll; the DLL will be loaded if it is present on the system.

Syntax

C++

```
HRESULT CreateDXGIFactory2(
    UINT    Flags,
    REFIID riid,
    [out] void    **ppFactory
);
```

Parameters

Flags

Type: **UINT**

Valid values include the **DXGI_CREATE_FACTORY_DEBUG** (0x01) flag, and zero.

Note This flag will be set by the D3D runtime if:

- The system creates an implicit factory during device creation.
- The **D3D11_CREATE_DEVICE_DEBUG** flag is specified during device creation, for example using **D3D11CreateDevice** (or the swapchain method, or the Direct3D 10 equivalents).

riid

Type: **REFIID**

The globally unique identifier (GUID) of the [IDXGIFactory2](#) object referenced by the *ppFactory* parameter.

[out] *ppFactory*

Type: **void****

Address of a pointer to an [IDXGIFactory2](#) object.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

This function accepts a flag indicating whether DXGI Debug.dll is loaded. The function otherwise behaves identically to [CreateDXGIFactory1](#).

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	DXGI.lib
DLL	Dxgi.dll

See also

[DXGI Functions](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_DECODE_SWAP_CHAIN_DESC structure (dxgi1_3.h)

Article04/02/2021

Used with [IDXGIFactoryMedia::CreateDecodeSwapChainForCompositionSurfaceHandle](#) to describe a decode swap chain.

Syntax

C++

```
typedef struct DXGI_DECODE_SWAP_CHAIN_DESC {
    UINT Flags;
} DXGI_DECODE_SWAP_CHAIN_DESC;
```

Members

Flags

Type: [UINT](#)

Can be 0, or a combination of [DXGI_SWAP_CHAIN_FLAG_FULLSCREEN_VIDEO](#) and/or [DXGI_SWAP_CHAIN_FLAG_YUV_VIDEO](#). Those named values are members of the [DXGI_SWAP_CHAIN_FLAG](#) enumerated type, and you can combine them by using a bitwise OR operation. The resulting value specifies options for decode swap-chain behavior.

Requirements

Header	dxgi1_3.h
--------	-----------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_FRAME_PRESENTATION_MODE enumeration (dxgi1_3.h)

Article01/31/2022

Indicates options for presenting frames to the swap chain.

Syntax

C++

```
typedef enum DXGI_FRAME_PRESENTATION_MODE {
    DXGI_FRAME_PRESENTATION_MODE_COMPOSED = 0,
    DXGI_FRAME_PRESENTATION_MODE_OVERLAY = 1,
    DXGI_FRAME_PRESENTATION_MODE_NONE = 2,
    DXGI_FRAME_PRESENTATION_MODE_COMPOSITION_FAILURE = 3
};
```

Constants

`DXGI_FRAME_PRESENTATION_MODE_COMPOSED`

Value: 0

Specifies that the presentation mode is a composition surface, meaning that the conversion from YUV to RGB is happening once per output refresh (for example, 60 Hz).

When this value is returned, the media app should discontinue use of the decode swap chain and perform YUV to RGB conversion itself, reducing the frequency of YUV to RGB conversion to once per video frame.

`DXGI_FRAME_PRESENTATION_MODE_OVERLAY`

Value: 1

Specifies that the presentation mode is an overlay surface, meaning that the YUV to RGB conversion is happening efficiently in hardware (once per video frame).

When this value is returned, the media app can continue to use the decode swap chain.

See [IDXGIDecodeSwapChain](#).

`DXGI_FRAME_PRESENTATION_MODE_NONE`

Value: 2

No presentation is specified.

`DXGI_FRAME_PRESENTATION_MODE_COMPOSITION_FAILURE`

Value: 3

An issue occurred that caused content protection to be invalidated in a swap-chain with hardware content protection, and is usually because the system ran out of hardware protected memory. The app will need to do one of the following:

- Drastically reduce the amount of hardware protected memory used. For example, media applications might be able to reduce their buffering.
- Stop using hardware protection if possible.

Note that simply re-creating the swap chain or the device will usually have no impact as the DWM will continue to run out of memory and will return the same failure.

Remarks

This enum is used by the [DXGI_FRAME_STATISTICS_MEDIA](#) structure.

Requirements

Header	dxgi1_3.h (include DXGIPartner.h)
--------	-----------------------------------

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_FRAME_STATISTICS_MEDIA structure (dxgi1_3.h)

Article04/02/2021

Used to verify system approval for the app's custom present duration (custom refresh rate). Approval should be continuously verified on a frame-by-frame basis.

Syntax

C++

```
typedef struct DXGI_FRAME_STATISTICS_MEDIA {
    UINT PresentCount;
    UINT PresentRefreshCount;
    UINT SyncRefreshCount;
    LARGE_INTEGER SyncQPCTime;
    LARGE_INTEGER SyncGPUMTime;
    DXGI_FRAME_PRESENTATION_MODE CompositionMode;
    UINT ApprovedPresentDuration;
} DXGI_FRAME_STATISTICS_MEDIA;
```

Members

PresentCount

Type: [UINT](#)

A value that represents the running total count of times that an image was presented to the monitor since the computer booted.

Note The number of times that an image was presented to the monitor is not necessarily the same as the number of times that you called [IDXGISwapChain::Present](#) or [IDXGISwapChain1::Present1](#).

PresentRefreshCount

Type: [UINT](#)

A value that represents the running total count of v-blanks at which the last image was presented to the monitor and that have happened since the computer booted (for

windowed mode, since the swap chain was created).

SyncRefreshCount

Type: **UINT**

A value that represents the running total count of v-blanks when the scheduler last sampled the machine time by calling [QueryPerformanceCounter](#) and that have happened since the computer booted (for windowed mode, since the swap chain was created).

SyncQPCTime

Type: **LARGE_INTEGER**

A value that represents the high-resolution performance counter timer. This value is the same as the value returned by the [QueryPerformanceCounter](#) function.

SyncGPUMTime

Type: **LARGE_INTEGER**

Reserved. Always returns 0.

CompositionMode

Type: **DXGI_FRAME_PRESENTATION_MODE**

A value indicating the composition presentation mode. This value is used to determine whether the app should continue to use the decode swap chain. See [DXGI_FRAME_PRESENTATION_MODE](#).

ApprovedPresentDuration

Type: **UINT**

If the system approves an app's custom present duration request, this field is set to the approved custom present duration.

If the app's custom present duration request is not approved, this field is set to zero.

Remarks

This structure is used with the [GetFrameStatisticsMedia](#) method.

Requirements

Header	dxgi1_3.h
--------	-----------

See also

[DXGI Structures](#)

[IDXGISwapChainMedia](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_MATRIX_3X2_F structure (dxgi1_3.h)

Article04/02/2021

Represents a 3x2 matrix. Used with [GetMatrixTransform](#) and [SetMatrixTransform](#) to indicate the scaling and translation transform for [SwapChainPanel](#) swap chains.

Syntax

C++

```
typedef struct DXGI_MATRIX_3X2_F {
    FLOAT _11;
    FLOAT _12;
    FLOAT _21;
    FLOAT _22;
    FLOAT _31;
    FLOAT _32;
} DXGI_MATRIX_3X2_F;
```

Members

_11

The value in the first row and first column of the matrix.

_12

The value in the first row and second column of the matrix.

_21

The value in the second row and first column of the matrix.

_22

The value in the second row and second column of the matrix.

_31

The value in the third row and first column of the matrix.

_32

The value in the third row and second column of the matrix.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Header	dxgi1_3.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_MULTIPLANE_OVERLAY_YCbCr_FL AGS enumeration (dxgi1_3.h)

Article11/03/2022

Options for swap-chain color space.

Syntax

C++

```
typedef enum DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS {
    DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_NOMINAL_RANGE = 0x1,
    DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_BT709 = 0x2,
    DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_xvYCC = 0x4
} ;
```

Constants

`DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_NOMINAL_RANGE`

Value: `0x1`

Specifies nominal range YCbCr, which isn't an absolute color space, but a way of encoding RGB info.

`DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_BT709`

Value: `0x2`

Specifies BT.709, which standardizes the format of high-definition television and has 16:9 (widescreen) aspect ratio.

`DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAG_xvYCC`

Value: `0x4`

Specifies xvYCC or extended-gamut YCC (also x.v.Color) color space that can be used in the video electronics of television sets to support a gamut 1.8 times as large as that of the sRGB color space.

Remarks

This enum is used by [SetColorSpace](#).

Requirements

Header

dxgi1_3.h (include DXGIPartner.h)

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OVERLAY_SUPPORT_FLAG enumeration (dxgi1_3.h)

Article01/31/2022

Specifies overlay support to check for in a call to [IDXGIOutput3::CheckOverlaySupport](#).

Syntax

C++

```
typedef enum DXGI_OVERLAY_SUPPORT_FLAG {  
    DXGI_OVERLAY_SUPPORT_FLAG_DIRECT = 0x1,  
    DXGI_OVERLAY_SUPPORT_FLAG_SCALING = 0x2  
} ;
```

Constants

`DXGI_OVERLAY_SUPPORT_FLAG_DIRECT`

Value: `0x1`

Direct overlay support.

`DXGI_OVERLAY_SUPPORT_FLAG_SCALING`

Value: `0x2`

Scaling overlay support.

Requirements

Minimum supported client Windows 8.1 [desktop apps only]

Minimum supported server Windows Server 2012 R2 [desktop apps only]

Header dxgi1_3.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGIGetDebugInterface1 function (dxgi1_3.h)

Article 10/13/2021

Retrieves an interface that Windows Store apps use for debugging the Microsoft DirectX Graphics Infrastructure (DXGI).

Syntax

C++

```
HRESULT DXGIGetDebugInterface1(
    UINT    Flags,
    REFIID riid,
    [out] void    **pDebug
);
```

Parameters

Flags

Not used.

riid

The globally unique identifier (GUID) of the requested interface type, which can be the identifier for the [IDXGIDebug](#), [IDXGIDebug1](#), or [IDXGILInfoQueue](#) interfaces.

[out] pDebug

A pointer to a buffer that receives a pointer to the debugging interface.

Return value

If this function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Remarks

The [DXGIGetDebugInterface1](#) function returns [E_NOINTERFACE](#) on systems without the Windows Software Development Kit (SDK) installed, because it's a development-time

aid.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	DXGI.lib
DLL	Dxgi.dll

See also

[DXGI Functions](#)

[IDXGIDebug1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain interface (dxgi1_3.h)

Article 07/27/2022

Represents a swap chain that is used by desktop media apps to decode video data and show it on a [DirectComposition](#) surface.

Inheritance

The **IDXGIDecodeSwapChain** interface inherits from the [IUnknown](#) interface.

IDXGIDecodeSwapChain also has these types of members:

Methods

The **IDXGIDecodeSwapChain** interface has these methods.

IDXGIDecodeSwapChain::GetColorSpace
Gets the color space used by the swap chain.
IDXGIDecodeSwapChain::GetDestSize
Gets the size of the destination surface to use for the video processing blit operation.
IDXGIDecodeSwapChain::GetSourceRect
Gets the source region that is used for the swap chain.
IDXGIDecodeSwapChain::GetTargetRect
Gets the rectangle that defines the target region for the video processing blit operation.
IDXGIDecodeSwapChain::PresentBuffer
Presents a frame on the output adapter.
IDXGIDecodeSwapChain::SetColorSpace
Sets the color space used by the swap chain. (<code>IDXGIDecodeSwapChain.SetColorSpace</code>)

[IDXGIDecodeSwapChain::SetDestSize](#)

Sets the size of the destination surface to use for the video processing blit operation.

[IDXGIDecodeSwapChain::SetSourceRect](#)

Sets the rectangle that defines the source region for the video processing blit operation.

[IDXGIDecodeSwapChain::SetTargetRect](#)

Sets the rectangle that defines the target region for the video processing blit operation.

Remarks

Decode swap chains are intended for use primarily with YUV surface formats. When using decode buffers created with an RGB surface format, the *TargetRect* and *DestSize* must be set equal to the buffer dimensions. *SourceRect* cannot exceed the buffer dimensions.

In clone mode, the decode swap chain is only guaranteed to be shown on the primary output.

Decode swap chains cannot be used with dirty rects.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIFactoryMedia](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::GetColorSpace method (dxgi1_3.h)

Article 06/29/2021

Gets the color space used by the swap chain.

Syntax

C++

```
DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS GetColorSpace();
```

Return value

A combination of [DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies the color space for the swap chain.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIDecodeSwapChain::GetDestSize method (dxgi1_3.h)

Article 10/13/2021

Gets the size of the destination surface to use for the video processing blit operation.

Syntax

C++

```
HRESULT GetDestSize(  
    [out] UINT *pWidth,  
    [out] UINT *pHeight  
>;
```

Parameters

[out] pWidth

A pointer to a variable that receives the width in pixels.

[out] pHeight

A pointer to a variable that receives the height in pixels.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows

Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDeclareSwapChain](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::GetSourceRect method (dxgi1_3.h)

Article 10/13/2021

Gets the source region that is used for the swap chain.

Syntax

C++

```
HRESULT GetSourceRect(  
    [out] RECT *pRect  
);
```

Parameters

[out] pRect

A pointer to a [RECT](#) structure that receives the source region for the swap chain.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::GetTargetRect method (dxgi1_3.h)

Article 10/13/2021

Gets the rectangle that defines the target region for the video processing blit operation.

Syntax

C++

```
HRESULT GetTargetRect(  
    [out] RECT *pRect  
);
```

Parameters

[out] pRect

A pointer to a [RECT](#) structure that receives the target region for the swap chain.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::PresentBuffer method (dxgi1_3.h)

Article 08/03/2021

Presents a frame on the output adapter. The frame is a subresource of the [IDXGIResource](#) object that was used to create the decode swap chain.

Syntax

C++

```
HRESULT PresentBuffer(  
    UINT BufferToPresent,  
    UINT SyncInterval,  
    UINT Flags  
>;
```

Parameters

BufferToPresent

An index indicating which member of the subresource array to present.

SyncInterval

An integer that specifies how to synchronize presentation of a frame with the vertical blank.

For the bit-block transfer (bitblt) model ([DXGI_SWAP_EFFECT_DISCARD](#) or [DXGI_SWAP_EFFECT_SEQUENTIAL](#)), values are:

- 0 - The presentation occurs immediately, there is no synchronization.
- 1,2,3,4 - Synchronize presentation after the *n*th vertical blank.

For the flip model ([DXGI_SWAP_EFFECT_FLIP_SEQUENTIAL](#)), values are:

- 0 - Cancel the remaining time on the previously presented frame and discard this frame if a newer frame is queued.
- *n* > 0 - Synchronize presentation for at least *n* vertical blanks.

Flags

An integer value that contains swap-chain presentation options. These options are defined by the [DXGI_PRESENT](#) constants.

The [DXGI_PRESENT_USE_DURATION](#) flag must be set if a custom present duration (custom refresh rate) is being used.

Return value

This method returns [S_OK](#) on success, or it returns one of the following error codes:

- [DXGI_ERROR_DEVICE_REMOVED](#)
- [DXGI_STATUS_OCCLUDED](#)
- [DXGI_ERROR_INVALID_CALL](#)
- [E_OUTOFMEMORY](#)

Requirements

Minimum supported client	
	Windows 8.1 [desktop apps only]
Minimum supported server	
	Windows Server 2012 R2 [desktop apps only]
Target Platform	
	Windows
Header	
	dxgi1_3.h
Library	
	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::SetColorSpace method (dxgi1_3.h)

Article07/27/2022

Sets the color space used by the swap chain.

Syntax

C++

```
HRESULT SetColorSpace(  
    DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS ColorSpace  
);
```

Parameters

`ColorSpace`

A pointer to a combination of [DXGI_MULTIPLANE_OVERLAY_YCbCr_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies the color space to set for the swap chain.

Return value

This method returns `S_OK` on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::SetDestSize method (dxgi1_3.h)

Article04/02/2021

Sets the size of the destination surface to use for the video processing blit operation.

The destination rectangle is the portion of the output surface that receives the blit for this stream. The destination rectangle is given in pixel coordinates, relative to the output surface.

Syntax

C++

```
HRESULT SetDestSize(  
    UINT Width,  
    UINT Height  
>;
```

Parameters

Width

The width of the destination size, in pixels.

Height

The height of the destination size, in pixels.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client

Windows 8.1 [desktop apps only]

Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDecodeSwapChain](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::SetSourceRect method (dxgi1_3.h)

Article04/02/2021

Sets the rectangle that defines the source region for the video processing blit operation.

The source rectangle is the portion of the input surface that is blitted to the destination surface. The source rectangle is given in pixel coordinates, relative to the input surface.

Syntax

C++

```
HRESULT SetSourceRect(  
    const RECT *pRect  
) ;
```

Parameters

pRect

A pointer to a [RECT](#) structure that contains the source region to set for the swap chain.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

Library

Dxgi.lib

See also

[IDXGIDeleteSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDecodeSwapChain::SetTargetRect method (dxgi1_3.h)

Article04/02/2021

Sets the rectangle that defines the target region for the video processing blit operation.

The target rectangle is the area within the destination surface where the output will be drawn. The target rectangle is given in pixel coordinates, relative to the destination surface.

Syntax

C++

```
HRESULT SetTargetRect(  
    const RECT *pRect  
>;
```

Parameters

pRect

A pointer to a [RECT](#) structure that contains the target region to set for the swap chain.

Return value

This method returns S_OK on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

Library

Dxgi.lib

See also

[IDXGIDeleteSwapChain](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice3 interface (dxgi1_3.h)

Article07/22/2021

The **IDXGIDevice3** interface implements a derived class for DXGI objects that produce image data. The interface exposes a method to trim graphics memory usage by the DXGI device.

Inheritance

The **IDXGIDevice3** interface inherits from [IDXGIDevice2](#). **IDXGIDevice3** also has these types of members:

Methods

The **IDXGIDevice3** interface has these methods.

[IDXGIDevice3::Trim](#)

Trims the graphics memory allocated by the **IDXGIDevice3** DXGI device on the app's behalf.

Remarks

The **IDXGIDevice3** interface is designed for use by DXGI objects that need access to other DXGI objects. This interface is useful to applications that do not use Direct3D to communicate with DXGI.

The Direct3D create device functions return a Direct3D device object. This Direct3D device object implements the [IUnknown](#) interface. You can query this Direct3D device object for the device's corresponding **IDXGIDevice3** interface. To retrieve the **IDXGIDevice3** interface of a Direct3D device, use the following code:

C++

```
IDXGIDevice3 * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice3), (void  
**)&pDXGIDevice);
```

Windows Phone 8: This API is supported.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIDevice2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice3::Trim method (dxgi1_3.h)

Article06/29/2021

Trims the graphics memory allocated by the [IDXGIDevice3](#) DXGI device on the app's behalf.

For apps that render with DirectX, graphics drivers periodically allocate internal memory buffers in order to speed up subsequent rendering requests. These memory allocations count against the app's memory usage for PLM and in general lead to increased memory usage by the overall system.

Starting in Windows 8.1, apps that render with Direct2D and/or Direct3D (including [CoreWindow](#) and XAML interop) must call **Trim** in response to the PLM suspend callback. The Direct3D runtime and the graphics driver will discard internal memory buffers allocated for the app, reducing its memory footprint.

Calling this method does not change the rendering state of the graphics device and it has no effect on rendering operations. There is a brief performance hit when internal buffers are reallocated during the first rendering operations after the **Trim** call, therefore apps should only call **Trim** when going idle for a period of time (in response to PLM suspend, for example).

Apps should ensure that they call **Trim** as one of the last D3D operations done before going idle. Direct3D will normally defer the destruction of D3D objects. Calling **Trim**, however, forces Direct3D to destroy objects immediately. For this reason, it is not guaranteed that releasing the final reference on Direct3D objects after calling **Trim** will cause the object to be destroyed and memory to be deallocated before the app suspends.

Similar to [ID3D11DeviceContext::Flush](#), apps should call [ID3D11DeviceContext::ClearState](#) before calling **Trim**. **ClearState** clears the Direct3D pipeline bindings, ensuring that Direct3D does not hold any references to the Direct3D objects you are trying to release.

It is also prudent to release references on middleware before calling **Trim**, as that middleware may also need to release references to Direct3D objects.

Syntax

C++

```
void Trim();
```

Return value

None

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIDevice3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory3 interface (dxgi1_3.h)

Article07/27/2022

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects.

Inheritance

The **IDXGIFactory3** interface inherits from [IDXGIFactory2](#). **IDXGIFactory3** also has these types of members:

Methods

The **IDXGIFactory3** interface has these methods.

[IDXGIFactory3::GetCreationFlags](#)

Gets the flags that were used when a Microsoft DirectX Graphics Infrastructure (DXGI) object was created.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

[IDXGIFactory2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory3::GetCreationFlags method (dxgi1_3.h)

Article 06/29/2021

Gets the flags that were used when a Microsoft DirectX Graphics Infrastructure (DXGI) object was created.

Syntax

C++

```
UINT GetCreationFlags();
```

Return value

The creation flags.

Remarks

The `GetCreationFlags` method returns flags that were passed to the [CreateDXGIFactory2](#) function, or were implicitly constructed by [CreateDXGIFactory](#), [CreateDXGIFactory1](#), [D3D11CreateDevice](#), or [D3D11CreateDeviceAndSwapChain](#).

Requirements

Target Platform	Windows
Header	dxgi1_3.h

See also

[IDXGIFactory3](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIFactoryMedia interface (dxgi1_3.h)

Article07/27/2022

Creates swap chains for desktop media apps that use [DirectComposition](#) surfaces to decode and display video.

Inheritance

The **IDXGIFactoryMedia** interface inherits from the [IUnknown](#) interface.

IDXGIFactoryMedia also has these types of members:

Methods

The **IDXGIFactoryMedia** interface has these methods.

[IDXGIFactoryMedia::CreateDecodeSwapChainForCompositionSurfaceHandle](#)

Creates a YUV swap chain for an existing DirectComposition surface handle.
(`IDXGIFactoryMedia.CreateDecodeSwapChainForCompositionSurfaceHandle`)

[IDXGIFactoryMedia::CreateSwapChainForCompositionSurfaceHandle](#)

Creates a YUV swap chain for an existing DirectComposition surface handle.
(`IDXGIFactoryMedia.CreateSwapChainForCompositionSurfaceHandle`)

Remarks

To create a Microsoft DirectX Graphics Infrastructure (DXGI) media factory interface, pass **IDXGIFactoryMedia** into either the [CreateDXGIFactory](#) or [CreateDXGIFactory1](#) function or call [QueryInterface](#) from a factory object returned by [CreateDXGIFactory](#), [CreateDXGIFactory1](#), or [CreateDXGIFactory2](#).

Because you can create a Direct3D device without creating a swap chain, you might need to retrieve the factory that is used to create the device in order to create a swap chain. You can request the [IDXGIDevice](#), [IDXGIDevice1](#), [IDXGIDevice2](#), or [IDXGIDevice3](#) interface from the Direct3D device and then use the [IDXGIOBJECT::GetParent](#) method to locate the factory. The following code shows how.

C++

```
IDXGIDevice2 * pDXGIDevice;
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice2), (void **)&pDXGIDevice);

IDXGIAdapter * pDXGIAdapter;
hr = pDXGIDevice->GetParent(__uuidof(IDXGIAdapter), (void **)&pDXGIAdapter);

IDXGIFactoryMedia * pIDXGIFactory;
pDXGIAdapter->GetParent(__uuidof(IDXGIFactoryMedia), (void **)&pIDXGIFactory);
```

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[DirectComposition](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactoryMedia::CreateDecodeSwapChainForCompositionSurfaceHandle method (dxgi1_3.h)

Article 07/27/2022

Creates a YUV swap chain for an existing [DirectComposition](#) surface handle. The swap chain is created with pre-existing buffers and very few descriptive elements are required. Instead, this method requires a [DirectComposition](#) surface handle and an [IDXGIResource](#) buffer to hold decoded frame data. The swap chain format is determined by the format of the subresources of the [IDXGIResource](#).

Syntax

C++

```
HRESULT CreateDecodeSwapChainForCompositionSurfaceHandle(
    [in]          IUnknown                      *pDevice,
    [in, optional] HANDLE                       hSurface,
    [in]          DXGI_DECODE_SWAP_CHAIN_DESC *pDesc,
    [in]          IDXGIResource                 *pYuvDecodeBuffers,
    [in, optional] IDXGIOOutput                  *pRestrictToOutput,
    [out]         IDXGIDecodeSwapChain        **ppSwapChain
);
```

Parameters

[in] pDevice

A pointer to the Direct3D device for the swap chain. This parameter cannot be **NULL**. Software drivers, like [D3D_DRIVER_TYPE_REFERENCE](#), are not supported for composition swap chains.

[in, optional] hSurface

A handle to an existing [DirectComposition](#) surface. This parameter cannot be **NULL**.

[in] pDesc

A pointer to a [DXGI_DECODE_SWAP_CHAIN_DESC](#) structure for the swap-chain description. This parameter cannot be **NULL**.

[in] *pYuvDecodeBuffers*

A pointer to a [IDXGIResource](#) interface that represents the resource that contains the info that [CreateDecodeSwapChainForCompositionSurfaceHandle](#) decodes.

[in, optional] *pRestrictToOutput*

A pointer to the [IDXGIOutput](#) interface for the swap chain to restrict content to. If the swap chain is moved to a different output, the content is black. You can optionally set this parameter to an output target that uses [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) to restrict the content on this output. If the swap chain is moved to a different output, the content is black.

You must also pass the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag in a present call to force the content to appear blacked out on any other output. If you want to restrict the content to a different output, you must create a new swap chain. However, you can conditionally restrict content based on the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag.

Set this parameter to **NULL** if you don't want to restrict content to an output target.

[out] *ppSwapChain*

A pointer to a variable that receives a pointer to the [IDXGIDecodeSwapChain](#) interface for the swap chain that this method creates.

Return value

[CreateDecodeSwapChainForCompositionSurfaceHandle](#) returns:

- S_OK if it successfully created a swap chain.
- E_OUTOFMEMORY if memory is unavailable to complete the operation.
- [DXGI_ERROR_INVALID_CALL](#) if the calling application provided invalid data, for example, if *pDesc*, *pYuvDecodeBuffers*, or *ppSwapChain* is **NULL**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic that are defined by the type of device that you pass to *pDevice*.

Remarks

The [IDXGIResource](#) provided via the *pYuvDecodeBuffers* parameter must point to at least one subresource, and all subresources must be created with the [D3D11_BIND_DECODER](#) flag.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIFactoryMedia](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactoryMedia::CreateSwapChainForCompositionSurfaceHandle method (dxgi1_3.h)

Article 07/27/2022

Creates a YUV swap chain for an existing [DirectComposition](#) surface handle.

Syntax

C++

```
HRESULT CreateSwapChainForCompositionSurfaceHandle(
    [in]           IUnknown                      *pDevice,
    [in, optional] HANDLE                        hSurface,
    [in]           const DXGI_SWAP_CHAIN_DESC1 *pDesc,
    [in, optional] IDXGIOOutput                 *pRestrictToOutput,
    [out]          IDXGISwapChain1              **ppSwapChain
);
```

Parameters

[in] pDevice

A pointer to the Direct3D device for the swap chain. This parameter cannot be **NULL**. Software drivers, like [D3D_DRIVER_TYPE_REFERENCE](#), are not supported for composition swap chains.

[in, optional] hSurface

A handle to an existing [DirectComposition](#) surface. This parameter cannot be **NULL**.

[in] pDesc

A pointer to a [DXGI_SWAP_CHAIN_DESC1](#) structure for the swap-chain description. This parameter cannot be **NULL**.

[in, optional] pRestrictToOutput

A pointer to the [IDXGIOOutput](#) interface for the swap chain to restrict content to. If the swap chain is moved to a different output, the content is black. You can optionally set this parameter to an output target that uses [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) to

restrict the content on this output. If the swap chain is moved to a different output, the content is black.

You must also pass the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag in a present call to force the content to appear blacked out on any other output. If you want to restrict the content to a different output, you must create a new swap chain. However, you can conditionally restrict content based on the [DXGI_PRESENT_RESTRICT_TO_OUTPUT](#) flag.

Set this parameter to **NULL** if you don't want to restrict content to an output target.

[out] ppSwapChain

A pointer to a variable that receives a pointer to the [IDXGISwapChain1](#) interface for the swap chain that this method creates.

Return value

CreateSwapChainForCompositionSurfaceHandle returns:

- **S_OK** if it successfully created a swap chain.
- **E_OUTOFMEMORY** if memory is unavailable to complete the operation.
- [DXGI_ERROR_INVALID_CALL](#) if the calling application provided invalid data, for example, if *pDesc*, *pYuvDecodeBuffers*, or *ppSwapChain* is **NULL**.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic that are defined by the type of device that you pass to *pDevice*.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[For best performance, use DXGI flip model](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIOutput2 interface (dxgi1_3.h)

Article07/22/2021

Represents an adapter output (such as a monitor). The **IDXGIOutput2** interface exposes a method to check for multiplane overlay support on the primary output adapter.

Inheritance

The **IDXGIOutput2** interface inherits from [IDXGIOutput1](#). **IDXGIOutput2** also has these types of members:

Methods

The **IDXGIOutput2** interface has these methods.

[IDXGIOutput2::SupportsOverlays](#)

Queries an adapter output for multiplane overlay support.

Remarks

To determine the outputs that are available from the adapter, use [IDXGIApapter::EnumOutputs](#). To determine the specific output that the swap chain will update, use [IDXGISwapChain::GetContainingOutput](#). You can then call [QueryInterface](#) from any [IDXGIOutput](#) or [IDXGIOutput1](#) object to obtain an **IDXGIOutput2** object.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIOutput1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput2::SupportsOverlays method (dxgi1_3.h)

Article 06/29/2021

Queries an adapter output for multiplane overlay support. If this API returns 'TRUE', multiple swap chain composition takes place in a performant manner using overlay hardware. If this API returns false, apps should avoid using foreground swap chains (that is, avoid using swap chains created with the [DXGI_SWAP_CHAIN_FLAG_FOREGROUND_LAYER](#) flag).

Syntax

C++

```
BOOL SupportsOverlays();
```

Return value

TRUE if the output adapter is the primary adapter and it supports multiplane overlays, otherwise returns FALSE.

Remarks

See [CreateSwapChainForCoreWindow](#) for info on creating a foreground swap chain.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIOutput2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput3 interface (dxgi1_3.h)

Article07/22/2021

Represents an adapter output (such as a monitor). The **IDXGIOutput3** interface exposes a method to check for overlay support.

Inheritance

The **IDXGIOutput3** interface inherits from [IDXGIOutput2](#). **IDXGIOutput3** also has these types of members:

Methods

The **IDXGIOutput3** interface has these methods.

[IDXGIOutput3::CheckOverlaySupport](#)

Checks for overlay support.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIOutput2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput3::CheckOverlaySupport method (dxgi1_3.h)

Article 10/13/2021

Checks for overlay support.

Syntax

C++

```
HRESULT CheckOverlaySupport(
    [in] DXGI_FORMAT EnumFormat,
    [in] IUnknown     *pConcernedDevice,
    [out] UINT        *pFlags
);
```

Parameters

[in] `EnumFormat`

Type: [DXGI_FORMAT](#)

A [DXGI_FORMAT](#)-typed value for the color format.

[in] `pConcernedDevice`

Type: [IUnknown*](#)

A pointer to the Direct3D device interface. **CheckOverlaySupport** returns only support info about this scan-out device.

[out] `pFlags`

Type: [UINT*](#)

A pointer to a variable that receives a combination of [DXGI_OVERLAY_SUPPORT_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for overlay support.

Return value

Type: [HRESULT](#)

Returns one of the error codes described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGIOutput3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2 interface (dxgi1_3.h)

Article 07/22/2021

Extends [IDXGISwapChain1](#) with methods to support swap back buffer scaling and lower-latency swap chains.

Inheritance

The [IDXGISwapChain2](#) interface inherits from [IDXGISwapChain1](#). [IDXGISwapChain2](#) also has these types of members:

Methods

The [IDXGISwapChain2](#) interface has these methods.

IDXGISwapChain2::GetFrameLatencyWaitableObject
Returns a waitable handle that signals when the DXGI adapter has finished presenting a new frame.
IDXGISwapChain2::GetMatrixTransform
Gets the transform matrix that will be applied to a composition swap chain upon the next present.
IDXGISwapChain2::GetMaximumFrameLatency
Gets the number of frames that the swap chain is allowed to queue for rendering.
IDXGISwapChain2::GetSourceSize
Gets the source region used for the swap chain.
IDXGISwapChain2::SetMatrixTransform
Sets the transform matrix that will be applied to a composition swap chain upon the next present.
IDXGISwapChain2::SetMaximumFrameLatency
Sets the number of frames that the swap chain is allowed to queue for rendering.
IDXGISwapChain2::SetSourceSize
Sets the source region to be used for the swap chain.

Remarks

You can create a swap chain by calling [IDXGIFactory2::CreateSwapChainForHwnd](#), [IDXGIFactory2::CreateSwapChainForCoreWindow](#), or [IDXGIFactory2::CreateSwapChainForComposition](#).

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIFactory2::CreateSwapChainForComposition](#)

[IDXGIFactory2::CreateSwapChainForCoreWindow](#)

[IDXGIFactory2::CreateSwapChainForHwnd](#)

[IDXGISwapChain1](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::GetFrameLatencyWaitableObject method (dxgi1_3.h)

Article 06/29/2021

Returns a waitable handle that signals when the DXGI adapter has finished presenting a new frame.

Windows 8.1 introduces new APIs that allow lower-latency rendering by waiting until the previous frame is presented to the display before drawing the next frame. To use this method, first create the DXGI swap chain with the

`DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT` flag set, then call `GetFrameLatencyWaitableObject` to retrieve the waitable handle. Use the waitable handle with `WaitForSingleObjectEx` to synchronize rendering of each new frame with the end of the previous frame. For every frame it renders, the app should wait on this handle before starting any rendering operations. Note that this requirement includes the first frame the app renders with the swap chain. See the [DirectXLatency sample](#). When you are done with the handle, use `CloseHandle` to close it.

Syntax

C++

```
HANDLE GetFrameLatencyWaitableObject();
```

Return value

A handle to the waitable object, or NULL if the swap chain was not created with `DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT`.

Remarks

When an application is finished using the object handle returned by `IDXGISwapChain2::GetFrameLatencyWaitableObject`, use the `CloseHandle` function to close the handle.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[DirectX latency sample](#) ↗

[GetMaximumFrameLatency](#)

[IDXGISwapChain2](#)

[SetMaximumFrameLatency](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::GetMatrixTransform method (dxgi1_3.h)

Article04/02/2021

Gets the transform matrix that will be applied to a composition swap chain upon the next present.

Starting with Windows 8.1, Windows Store apps are able to place DirectX swap chain visuals in XAML pages using the [SwapChainPanel](#) element, which can be placed and sized arbitrarily. This exposes the DirectX swap chain visuals to touch scaling and translation scenarios using touch UI. The [GetMatrixTransform](#) and [SetMatrixTransform](#) methods are used to synchronize scaling of the DirectX swap chain with its associated [SwapChainPanel](#) element. Only simple scale/translation elements in the matrix are allowed – the call will fail if the matrix contains skew/rotation elements.

Syntax

C++

```
HRESULT GetMatrixTransform(
    DXGI_MATRIX_3X2_F *pMatrix
);
```

Parameters

pMatrix

[out]

The transform matrix currently used for swap chain scaling and translation.

Return value

[GetMatrixTransform](#) returns:

- S_OK if it successfully retrieves the transform matrix.
- DXGI_ERROR_INVALID_CALL if the method is called on a swap chain that was not created with [CreateSwapChainForComposition](#).
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGISwapChain2](#)

[SetMatrixTransform](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::GetMaximumFrameLatency method (dxgi1_3.h)

Article 10/13/2021

Gets the number of frames that the swap chain is allowed to queue for rendering.

Syntax

C++

```
HRESULT GetMaximumFrameLatency(
    [out] UINT *pMaxLatency
);
```

Parameters

[out] pMaxLatency

The maximum number of back buffer frames that will be queued for the swap chain. This value is 1 by default, but should be set to 2 if the scene takes longer than it takes for one vertical refresh (typically about 16ms) to draw.

Return value

Returns S_OK if successful; otherwise, returns one of the following members of the D3DERR enumerated type:

- D3DERR_DEVICELOST
- D3DERR_DEVICEREMOVED
- D3DERR_DRIVERINTERNALERROR
- D3DERR_INVALIDCALL
- D3DERR_OUTOFVIDEOMEMORY

Requirements

Minimum supported client

Windows 8.1 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[DirectX latency sample](#)

[IDXGISwapChain2](#)

[SetMaximumFrameLatency](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::GetSourceSize method (dxgi1_3.h)

Article 10/13/2021

Gets the source region used for the swap chain.

Use **GetSourceSize** to get the portion of the swap chain from which the operating system presents. The source rectangle is always defined by the region [0, 0, Width, Height]. Use [SetSourceSize](#) to set this portion of the swap chain.

Syntax

C++

```
HRESULT GetSourceSize(  
    [out] UINT *pWidth,  
    [out] UINT *pHeight  
);
```

Parameters

`[out] pWidth`

The current width of the source region of the swap chain. This value can range from 1 to the overall width of the swap chain.

`[out] pHeight`

The current height of the source region of the swap chain. This value can range from 1 to the overall height of the swap chain.

Return value

This method can return error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGISwapChain2](#)

[SetSourceSize](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::SetMatrixTransform method (dxgi1_3.h)

Article04/02/2021

Sets the transform matrix that will be applied to a composition swap chain upon the next present.

Starting with Windows 8.1, Windows Store apps are able to place DirectX swap chain visuals in XAML pages using the [SwapChainPanel](#) element, which can be placed and sized arbitrarily. This exposes the DirectX swap chain visuals to touch scaling and translation scenarios using touch UI. The [GetMatrixTransform](#) and [SetMatrixTransform](#) methods are used to synchronize scaling of the DirectX swap chain with its associated [SwapChainPanel](#) element. Only simple scale/translation elements in the matrix are allowed – the call will fail if the matrix contains skew/rotation elements.

Syntax

C++

```
HRESULT SetMatrixTransform(  
    const DXGI_MATRIX_3X2_F *pMatrix  
);
```

Parameters

pMatrix

The transform matrix to use for swap chain scaling and translation. This function can only be used with composition swap chains created by [IDXGIFactory2::CreateSwapChainForComposition](#). Only scale and translation components are allowed in the matrix.

Return value

[SetMatrixTransform](#) returns:

- S_OK if it successfully retrieves the transform matrix.
- E_INVALIDARG if the *pMatrix* parameter is incorrect, for example, *pMatrix* is NULL or the matrix represented by [DXGI_MATRIX_3X2_F](#) includes components other than

scale and translation.

- DXGI_ERROR_INVALID_CALL if the method is called on a swap chain that was not created with [CreateSwapChainForComposition](#).
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[GetMatrixTransform](#)

[IDXGISwapChain2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::SetMaximumFrameLatency method (dxgi1_3.h)

Article04/02/2021

Sets the number of frames that the swap chain is allowed to queue for rendering.

Syntax

C++

```
HRESULT SetMaximumFrameLatency(  
    UINT MaxLatency  
>;
```

Parameters

MaxLatency

The maximum number of back buffer frames that will be queued for the swap chain. This value is 1 by default.

Return value

Returns S_OK if successful; otherwise, DXGI_ERROR_DEVICE_REMOVED if the device was removed.

Remarks

This method is only valid for use on swap chains created with [DXGI_SWAP_CHAIN_FLAG_FRAME_LATENCY_WAITABLE_OBJECT](#). Otherwise, the result will be DXGI_ERROR_INVALID_CALL.

Requirements

Minimum supported client

Windows 8.1 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[DirectX latency sample](#)

[GetMaximumFrameLatency](#)

[IDXGISwapChain2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain2::SetSourceSize method (dxgi1_3.h)

Article04/02/2021

Sets the source region to be used for the swap chain.

Use **SetSourceSize** to specify the portion of the swap chain from which the operating system presents. This allows an effective resize without calling the more-expensive [IDXGISwapChain::ResizeBuffers](#) method. Prior to Windows 8.1, calling [IDXGISwapChain::ResizeBuffers](#) was the only way to resize the swap chain. The source rectangle is always defined by the region [0, 0, Width, Height].

Syntax

C++

```
HRESULT SetSourceSize(  
    UINT Width,  
    UINT Height  
>;
```

Parameters

Width

Source width to use for the swap chain. This value must be greater than zero, and must be less than or equal to the overall width of the swap chain.

Height

Source height to use for the swap chain. This value must be greater than zero, and must be less than or equal to the overall height of the swap chain.

Return value

This method can return:

- **E_INVALIDARG** if one or more parameters exceed the size of the back buffer.
- Possibly other error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[GetSourceSize](#)

[IDXGISwapChain2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChainMedia interface (dxgi1_3.h)

Article 07/22/2021

This swap chain interface allows desktop media applications to request a seamless change to a specific refresh rate.

For example, a media application presenting video at a typical framerate of 23.997 frames per second can request a custom refresh rate of 24 or 48 Hz to eliminate jitter. If the request is approved, the app starts presenting frames at the custom refresh rate immediately - without the typical 'mode switch' a user would experience when changing the refresh rate themselves by using the control panel.

Inheritance

The **IDXGISwapChainMedia** interface inherits from the [IUnknown](#) interface.
IDXGISwapChainMedia also has these types of members:

Methods

The **IDXGISwapChainMedia** interface has these methods.

IDXGISwapChainMedia::CheckPresentDurationSupport
Queries the graphics driver for a supported frame present duration corresponding to a custom refresh rate.
IDXGISwapChainMedia::GetFrameStatisticsMedia
Queries the system for a DXGI_FRAME_STATISTICS_MEDIA structure that indicates whether a custom refresh rate is currently approved by the system.
IDXGISwapChainMedia::SetPresentDuration
Requests a custom presentation duration (custom refresh rate).

Remarks

Seamless changes to custom fram rates can only be done on integrated panels. Custom frame rates cannot be applied to external displays. If the DXGI output adapter is attached to an external display then [CheckPresentDurationSupport](#) will return (0, 0) for upper and lower bounds, indicating that the device does not support seamless refresh rate changes.

Custom refresh rates can be used when displaying video with a dynamic framerate. However, the refresh rate change should be kept imperceptible to the user. A best practice for keeping the refresh rate transition imperceptible is to only set the custom framerate if the app determines it can present at that rate for least 5 seconds.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h

See also

[DXGI Interfaces](#)

[IDXGIFactoryMedia](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChainMedia::CheckPresentDurationSupport method (dxgi1_3.h)

Article 10/13/2021

Queries the graphics driver for a supported frame present duration corresponding to a custom refresh rate.

Syntax

C++

```
HRESULT CheckPresentDurationSupport(
    UINT DesiredPresentDuration,
    [out] UINT *pClosestSmallerPresentDuration,
    [out] UINT *pClosestLargerPresentDuration
);
```

Parameters

`DesiredPresentDuration`

Indicates the frame duration to check. This value is the duration of one frame at the desired refresh rate, specified in hundreds of nanoseconds. For example, set this field to 167777 to check for 60 Hz refresh rate support.

`[out] pClosestSmallerPresentDuration`

A variable that will be set to the closest supported frame present duration that's smaller than the requested value, or zero if the device does not support any lower duration.

`[out] pClosestLargerPresentDuration`

A variable that will be set to the closest supported frame present duration that's larger than the requested value, or zero if the device does not support any higher duration.

Return value

This method returns S_OK on success, or a DXGI error code on failure.

Remarks

If the DXGI output adapter does not support custom refresh rates (for example, an external display) then the display driver will set upper and lower bounds to (0, 0).

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGISwapChainMedia](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChainMedia::GetFrameStatisticsMedia method (dxgi1_3.h)

Article 10/13/2021

Queries the system for a [DXGI_FRAME_STATISTICS_MEDIA](#) structure that indicates whether a custom refresh rate is currently approved by the system.

Syntax

C++

```
HRESULT GetFrameStatisticsMedia(  
    [out] DXGI_FRAME_STATISTICS_MEDIA *pStats  
) ;
```

Parameters

[out] pStats

A [DXGI_FRAME_STATISTICS_MEDIA](#) structure indicating whether the system currently approves the custom refresh rate request.

Return value

This method returns S_OK on success, or a DXGI error code on failure.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

[IDXGISwapChainMedia](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChainMedia::SetPresentDuration method (dxgi1_3.h)

Article 04/02/2021

Requests a custom presentation duration (custom refresh rate).

Syntax

C++

```
HRESULT SetPresentDuration(  
    UINT Duration  
);
```

Parameters

Duration

The custom presentation duration, specified in hundreds of nanoseconds.

Return value

This method returns S_OK on success, or a DXGI error code on failure.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgi1_3.h
Library	Dxgi.lib

See also

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

dxgi1_4.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi1_4.h contains the following programming interfaces:

Interfaces

[IDXGIAdapter3](#)

This interface adds some memory residency methods, for budgeting and reserving physical memory.

[IDXGIFactory4](#)

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects. ([IDXGIFactory4](#))

[IDXGIOutput4](#)

Represents an adapter output (such as a monitor). The [IDXGIOutput4](#) interface exposes a method to check for overlay color space support.

[IDXGISwapChain3](#)

Extends [IDXGISwapChain2](#) with methods to support getting the index of the swap chain's current back buffer and support for color space.

Structures

[DXGI_QUERY_VIDEO_MEMORY_INFO](#)

Describes the current video memory budgeting parameters.

Enumerations

[DXGI_MEMORY_SEGMENT_GROUP](#)

Specifies the memory segment group to use.

[DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG](#)

Specifies support for overlay color space.

[DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG](#)

Specifies color space support for the swap chain.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_MEMORY_SEGMENT_GROUP

enumeration (dxgi1_4.h)

Article01/31/2022

Specifies the memory segment group to use.

Syntax

C++

```
typedef enum DXGI_MEMORY_SEGMENT_GROUP {  
    DXGI_MEMORY_SEGMENT_GROUP_LOCAL = 0,  
    DXGI_MEMORY_SEGMENT_GROUP_NON_LOCAL = 1  
};
```

Constants

`DXGI_MEMORY_SEGMENT_GROUP_LOCAL`

Value: 0

The grouping of segments which is considered local to the video adapter, and represents the fastest available memory to the GPU. Applications should target the local segment group as the target size for their working set.

`DXGI_MEMORY_SEGMENT_GROUP_NON_LOCAL`

Value: 1

The grouping of segments which is considered non-local to the video adapter, and may have slower performance than the local segment group.

Remarks

This enum is used by [QueryVideoMemoryInfo](#) and [SetVideoMemoryReservation](#).

Refer to the remarks for [D3D12_MEMORY_POOL](#).

Requirements

Header

dxgi1_4.h (include DXGI1_3.h)

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG enumeration (dxgi1_4.h)

Article01/31/2022

Specifies support for overlay color space.

Syntax

C++

```
typedef enum DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG {  
    DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG_PRESENT = 0x1  
} ;
```

Constants

`DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG_PRESENT`

Value: *0x1*

Overlay color space support is present.

Requirements

Minimum supported client

Windows 10 [desktop apps only]

Minimum supported server

Windows Server 2016 [desktop apps only]

Header

dxgi1_4.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_QUERY_VIDEO_MEMORY_INFO structure (dxgi1_4.h)

Article04/02/2021

Describes the current video memory budgeting parameters.

Syntax

C++

```
typedef struct DXGI_QUERY_VIDEO_MEMORY_INFO {
    UINT64 Budget;
    UINT64 CurrentUsage;
    UINT64 AvailableForReservation;
    UINT64 CurrentReservation;
} DXGI_QUERY_VIDEO_MEMORY_INFO;
```

Members

Budget

Specifies the OS-provided video memory budget, in bytes, that the application should target. If *CurrentUsage* is greater than *Budget*, the application may incur stuttering or performance penalties due to background activity by the OS to provide other applications with a fair usage of video memory.

CurrentUsage

Specifies the application's current video memory usage, in bytes.

AvailableForReservation

The amount of video memory, in bytes, that the application has available for reservation. To reserve this video memory, the application should call [IDXGIAAdapter3::SetVideoMemoryReservation](#).

CurrentReservation

The amount of video memory, in bytes, that is reserved by the application. The OS uses the reservation as a hint to determine the application's minimum working set.

Applications should attempt to ensure that their video memory usage can be trimmed to meet this requirement.

Remarks

Use this structure with [QueryVideoMemoryInfo](#).

Refer to the remarks for [D3D12_MEMORY_POOL](#).

Requirements

Header	dxgi1_4.h (include DXGI1_3.h)
--------	-------------------------------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG enumeration (dxgi1_4.h)

Article01/31/2022

Specifies color space support for the swap chain.

Syntax

C++

```
typedef enum DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG {
    DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG_PRESENT = 0x1,
    DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG_OVERLAY_PRESENT = 0x2
};
```

Constants

`DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG_PRESENT`

Value: `0x1`

Color space support is present.

`DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG_OVERLAY_PRESENT`

Value: `0x2`

Overlay color space support is present.

Requirements

Minimum supported client

Windows 10 [desktop apps only]

Minimum supported server

Windows Server 2016 [desktop apps only]

Header

dxgi1_4.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3 interface (dxgi1_4.h)

Article07/22/2021

This interface adds some memory residency methods, for budgeting and reserving physical memory.

Inheritance

The IDXGIAdapter3 interface inherits from IDXGIAdapter2. IDXGIAdapter3 also has these types of members:

Methods

The IDXGIAdapter3 interface has these methods.

IDXGIAdapter3::QueryVideoMemoryInfo
This method informs the process of the current budget and process usage.
IDXGIAdapter3::RegisterHardwareContentProtectionTeardownStatusEvent
Registers to receive notification of hardware content protection teardown events.
IDXGIAdapter3::RegisterVideoMemoryBudgetChangeNotificationEvent
This method establishes a correlation between a CPU synchronization object and the budget change event.
IDXGIAdapter3::SetVideoMemoryReservation
This method sends the minimum required physical memory for an application, to the OS.
IDXGIAdapter3::UnregisterHardwareContentProtectionTeardownStatus
Unregisters an event to stop it from receiving notification of hardware content protection teardown events.
IDXGIAdapter3::UnregisterVideoMemoryBudgetChangeNotification
This method stops notifying a CPU synchronization object whenever a budget change occurs. An application may switch back to polling the information regularly.

Remarks

For more details, refer to the [Residency](#) section of the D3D12 documentation.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)

See also

[DXGI Interfaces](#)

[IDXGIAAdapter2](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::QueryVideoMemoryInfo method (dxgi1_4.h)

Article 10/13/2021

This method informs the process of the current budget and process usage.

Syntax

C++

```
HRESULT QueryVideoMemoryInfo(
    [in]  UINT           NodeIndex,
    [in]  DXGI_MEMORY_SEGMENT_GROUP MemorySegmentGroup,
    [out] DXGI_QUERY_VIDEO_MEMORY_INFO *pVideoMemoryInfo
);
```

Parameters

[in] NodeIndex

Type: **UINT**

Specifies the device's physical adapter for which the video memory information is queried. For single-GPU operation, set this to zero. If there are multiple GPU nodes, set this to the index of the node (the device's physical adapter) for which the video memory information is queried. See [Multi-adapter systems](#).

[in] MemorySegmentGroup

Type: **DXGI_MEMORY_SEGMENT_GROUP**

Specifies a DXGI_MEMORY_SEGMENT_GROUP that identifies the group as local or non-local.

[out] pVideoMemoryInfo

Type: **DXGI_QUERY_VIDEO_MEMORY_INFO***

Fills in a DXGI_QUERY_VIDEO_MEMORY_INFO structure with the current values.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Applications must explicitly manage their usage of physical memory explicitly and keep usage within the budget assigned to the application process. Processes that cannot keep their usage within their assigned budgets will likely experience stuttering, as they are intermittently frozen and paged-out to allow other processes to run.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIAAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::RegisterHardwareContentProtectionTeardownStatusEvent method (dxgi1_4.h)

Article 10/13/2021

Registers to receive notification of hardware content protection teardown events.

Syntax

C++

```
HRESULT RegisterHardwareContentProtectionTeardownStatusEvent(
    [in]    HANDLE hEvent,
    [out]   DWORD  *pdwCookie
);
```

Parameters

[in] `hEvent`

Type: `HANDLE`

A handle to the event object that the operating system sets when hardware content protection teardown occurs. The [CreateEvent](#) or [OpenEvent](#) function returns this handle.

[out] `pdwCookie`

Type: `DWORD*`

A pointer to a key value that an application can pass to the [IDXGIAdapter3::UnregisterHardwareContentProtectionTeardownStatus](#) method to unregister the notification event that `hEvent` specifies.

Return value

Type: `HRESULT`

If this method succeeds, it returns `S_OK`. Otherwise, it returns an `HRESULT` error code.

Remarks

Call [ID3D11VideoDevice::GetContentProtectionCaps\(\)](#) to check for the presence of the **D3D11_CONTENT_PROTECTION_CAPS_HARDWARE_TEARDOWN** capability to know whether the hardware contains an automatic teardown mechanism.

After the event is signaled, the application can call [ID3D11VideoContext1::CheckCryptoSessionStatus](#) to determine the impact of the hardware teardown for a specific [ID3D11CryptoSession](#) interface.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::RegisterVideoMemoryBudgetChangeNotificationEvent method (dxgi1_4.h)

Article 10/13/2021

This method establishes a correlation between a CPU synchronization object and the budget change event.

Syntax

C++

```
HRESULT RegisterVideoMemoryBudgetChangeNotificationEvent(
    [in]    HANDLE hEvent,
    [out]   DWORD  *pdwCookie
);
```

Parameters

[in] `hEvent`

Type: `HANDLE`

Specifies a `HANDLE` for the event.

[out] `pdwCookie`

Type: `DWORD*`

A key value for the window or event to unregister. The [IDXGIAdapter3::RegisterHardwareContentProtectionTeardownStatusEvent](#) method returns this value.

Return value

Type: `HRESULT`

This method returns an `HRESULT` success or error code.

Remarks

Instead of calling [QueryVideoMemoryInfo](#) regularly, applications can use CPU synchronization objects to efficiently wake threads when budget changes occur.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::SetVideoMemoryReservation method (dxgi1_4.h)

Article 10/13/2021

This method sends the minimum required physical memory for an application, to the OS.

Syntax

C++

```
HRESULT SetVideoMemoryReservation(
    [in] UINT             NodeIndex,
    [in] DXGI_MEMORY_SEGMENT_GROUP MemorySegmentGroup,
    [in] UINT64           Reservation
);
```

Parameters

[in] NodeIndex

Type: **UINT**

Specifies the device's physical adapter for which the video memory information is being set. For single-GPU operation, set this to zero. If there are multiple GPU nodes, set this to the index of the node (the device's physical adapter) for which the video memory information is being set. See [Multi-adapter systems](#).

[in] MemorySegmentGroup

Type: [**DXGI_MEMORY_SEGMENT_GROUP**](#)

Specifies a DXGI_MEMORY_SEGMENT_GROUP that identifies the group as local or non-local.

[in] Reservation

Type: **UINT64**

Specifies a UINT64 that sets the minimum required physical memory, in bytes.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Applications are encouraged to set a video reservation to denote the amount of physical memory they cannot go without. This value helps the OS quickly minimize the impact of large memory pressure situations.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIAAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::UnregisterHardwareContentProtectionTeardownStatus method (dxgi1_4.h)

Article 10/13/2021

Unregisters an event to stop it from receiving notification of hardware content protection teardown events.

Syntax

C++

```
void UnregisterHardwareContentProtectionTeardownStatus(
    [in] DWORD dwCookie
);
```

Parameters

[in] dwCookie

Type: **DWORD**

A key value for the window or event to unregister. The [IDXGIAdapter3::RegisterHardwareContentProtectionTeardownStatusEvent](#) method returns this value.

Return value

None

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib

DLL

Dxgi.dll

See also

[IDXGIAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter3::UnregisterVideoMemoryBudgetChangeNotification method (dxgi1_4.h)

Article 10/13/2021

This method stops notifying a CPU synchronization object whenever a budget change occurs. An application may switch back to polling the information regularly.

Syntax

C++

```
void UnregisterVideoMemoryBudgetChangeNotification(
    [in] DWORD dwCookie
);
```

Parameters

[in] dwCookie

Type: **DWORD**

A key value for the window or event to unregister. The [IDXGIAdapter3::RegisterHardwareContentProtectionTeardownStatusEvent](#) method returns this value.

Return value

None

Remarks

An application may switch back to polling for the information regularly.

Requirements

Target Platform	Windows
Header	dxgi1_4.h (include DXGI1_3.h)
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIAAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory4 interface (dxgi1_4.h)

Article07/27/2022

Enables creating Microsoft DirectX Graphics Infrastructure (DXGI) objects.

Inheritance

The **IDXGIFactory4** interface inherits from [IDXGIFactory3](#). **IDXGIFactory4** also has these types of members:

Methods

The **IDXGIFactory4** interface has these methods.

IDXGIFactory4::EnumAdapterByLuid
Outputs the IDXGIAdapter for the specified LUID.
IDXGIFactory4::EnumWarpAdapter
Provides an adapter which can be provided to D3D12CreateDevice to use the WARP renderer.

Requirements

Target Platform	Windows
Header	dxgi1_4.h

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

[IDXGIFactory2](#)

[IDXGIFactory3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory4::EnumAdapterByLuid method (dxgi1_4.h)

Article 10/13/2021

Outputs the [IDXGIAdapter](#) for the specified LUID.

Syntax

C++

```
HRESULT EnumAdapterByLuid(
    [in] LUID AdapterLuid,
    [in] REFIID riid,
    [out] void    **ppvAdapter
);
```

Parameters

[in] AdapterLuid

Type: [LUID](#)

A unique value that identifies the adapter. See [LUID](#) for a definition of the structure.

LUID is defined in dxgi.h.

[in] riid

Type: [REFIID](#)

The globally unique identifier (GUID) of the [IDXGIAdapter](#) object referenced by the *ppvAdapter* parameter.

[out] ppvAdapter

Type: [void**](#)

The address of an [IDXGIAdapter](#) interface pointer to the adapter. This parameter must not be NULL.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#). See also Direct3D 12 Return Codes.

Remarks

For Direct3D 12, it's no longer possible to backtrack from a device to the [IDXGIAAdapter](#) that was used to create it. [IDXGIFactory4::EnumAdapterByLuid](#) enables an app to retrieve information about the adapter where a D3D12 device was created. [IDXGIFactory4::EnumAdapterByLuid](#) is designed to be paired with [ID3D12Device::GetAdapterLuid](#). For more information, see [DXGI 1.4 Improvements](#).

Requirements

Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory4::EnumWarpAdapter method (dxgi1_4.h)

Article 10/13/2021

Provides an adapter which can be provided to D3D12CreateDevice to use the WARP renderer.

Syntax

C++

```
HRESULT EnumWarpAdapter(
    [in] REFIID riid,
    [out] void    **ppvAdapter
);
```

Parameters

[in] *riid*

Type: **REFIID**

The globally unique identifier (GUID) of the [IDXGIAdapter](#) object referenced by the *ppvAdapter* parameter.

[out] *ppvAdapter*

Type: **void****

The address of an [IDXGIAdapter](#) interface pointer to the adapter. This parameter must not be NULL.

Return value

Type: **HRESULT**

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#). See also Direct3D 12 Return Codes.

Remarks

For more information, see [DXGI 1.4 Improvements](#).

Requirements

Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[DXGI Interfaces](#)

[IDXGIFactory4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput4 interface (dxgi1_4.h)

Article07/22/2021

Represents an adapter output (such as a monitor). The **IDXGIOutput4** interface exposes a method to check for overlay color space support.

Inheritance

The **IDXGIOutput4** interface inherits from [IDXGIOutput3](#). **IDXGIOutput4** also has these types of members:

Methods

The **IDXGIOutput4** interface has these methods.

[IDXGIOutput4::CheckOverlayColorSpaceSupport](#)

Checks for overlay color space support.

Requirements

Minimum supported client	Windows 10 [desktop apps UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_4.h

See also

[DXGI Interfaces](#)

[IDXGIOutput3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput4::CheckOverlayColorSpaceSupport method (dxgi1_4.h)

Article 10/13/2021

Checks for overlay color space support.

Syntax

C++

```
HRESULT CheckOverlayColorSpaceSupport(
    [in] DXGI_FORMAT           Format,
    [in] DXGI_COLOR_SPACE_TYPE ColorSpace,
    [in] IUnknown*             pConcernedDevice,
    [out] UINT*                pFlags
);
```

Parameters

[in] Format

Type: [DXGI_FORMAT](#)

A [DXGI_FORMAT](#)-typed value for the color format.

[in] ColorSpace

Type: [DXGI_COLOR_SPACE_TYPE](#)

A [DXGI_COLOR_SPACE_TYPE](#)-typed value that specifies color space type to check overlay support for.

[in] pConcernedDevice

Type: [IUnknown*](#)

A pointer to the Direct3D device interface. `CheckOverlayColorSpaceSupport` returns only support info about this scan-out device.

[out] pFlags

Type: [UINT*](#)

A pointer to a variable that receives a combination of [DXGI_OVERLAY_COLOR_SPACE_SUPPORT_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for overlay color space support.

Return value

Type: [HRESULT](#)

This method returns [S_OK](#) on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[IDXGIOutput4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain3 interface (dxgi1_4.h)

Article07/27/2022

Extends [IDXGISwapChain2](#) with methods to support getting the index of the swap chain's current back buffer and support for color space.

Inheritance

The [IDXGISwapChain3](#) interface inherits from [IDXGISwapChain2](#). [IDXGISwapChain3](#) also has these types of members:

Methods

The [IDXGISwapChain3](#) interface has these methods.

IDXGISwapChain3::CheckColorSpaceSupport
Checks the swap chain's support for color space.
IDXGISwapChain3::GetCurrentBackBufferIndex
Gets the index of the swap chain's current back buffer.
IDXGISwapChain3::ResizeBuffers1
Changes the swap chain's back buffer size, format, and number of buffers, where the swap chain was created using a D3D12 command queue as an input device. This should be called when the application window is resized.
IDXGISwapChain3::SetColorSpace1
Sets the color space used by the swap chain. (IDXGISwapChain3::SetColorSpace1)

Requirements

Minimum supported client	Windows 10 [desktop apps UWP apps]
Minimum supported server	Windows Server 2016 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgi1_4.h

See also

[DXGI Interfaces](#)

[IDXGISwapChain2](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGISwapChain3::CheckColorSpaceSupport method (dxgi1_4.h)

Article 10/13/2021

Checks the swap chain's support for color space.

Syntax

C++

```
HRESULT CheckColorSpaceSupport(
    [in]  DXGI_COLOR_SPACE_TYPE ColorSpace,
    [out] UINT                 *pColorSpaceSupport
);
```

Parameters

[in] ColorSpace

Type: [DXGI_COLOR_SPACE_TYPE](#)

A [DXGI_COLOR_SPACE_TYPE](#)-typed value that specifies color space type to check support for.

[out] pColorSpaceSupport

Type: [UINT*](#)

A pointer to a variable that receives a combination of [DXGI_SWAP_CHAIN_COLOR_SPACE_SUPPORT_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for color space support.

Return value

Type: [HRESULT](#)

This method returns [S_OK](#) on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[IDXGISwapChain3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain3::GetCurrentBackBufferIndex method (dxgi1_4.h)

Article 06/29/2021

Gets the index of the swap chain's current back buffer.

Syntax

C++

```
UINT GetCurrentBackBufferIndex();
```

Return value

Type: [UINT](#)

Returns the index of the current back buffer.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[IDXGISwapChain3](#)

Feedback



Was this page helpful?  

Get help at Microsoft Q&A

IDXGISwapChain3::ResizeBuffers1 method (dxgi1_4.h)

Article 05/24/2022

Changes the swap chain's back buffer size, format, and number of buffers, where the swap chain was created using a D3D12 command queue as an input device. This should be called when the application window is resized.

Syntax

C++

```
HRESULT ResizeBuffers1(
    [in] UINT          BufferCount,
    [in] UINT          Width,
    [in] UINT          Height,
    [in] DXGI_FORMAT   Format,
    [in] UINT          SwapChainFlags,
    [in] const UINT    *pCreationNodeMask,
    [in] IUnknown      * const *ppPresentQueue
);
```

Parameters

[in] BufferCount

Type: **UINT**

The number of buffers in the swap chain (including all back and front buffers). This number can be different from the number of buffers with which you created the swap chain. This number can't be greater than **DXGI_MAX_SWAP_CHAIN_BUFFERS**. Set this number to zero to preserve the existing number of buffers in the swap chain. You can't specify less than two buffers for the flip presentation model.

[in] Width

Type: **UINT**

The new width of the back buffer. If you specify zero, DXGI will use the width of the client area of the target window. You can't specify the width as zero if you called the [IDXGIFactory2::CreateSwapChainForComposition](#) method to create the swap chain for a composition surface.

[in] Height

Type: **UINT**

The new height of the back buffer. If you specify zero, DXGI will use the height of the client area of the target window. You can't specify the height as zero if you called the [IDXGIFactory2::CreateSwapChainForComposition](#) method to create the swap chain for a composition surface.

[in] Format

Type: **DXGI_FORMAT**

A [DXGI_FORMAT](#)-typed value for the new format of the back buffer. Set this value to [DXGI_FORMAT_UNKNOWN](#) to preserve the existing format of the back buffer. The flip presentation model supports a more restricted set of formats than the bit-block transfer (bitblt) model.

[in] SwapChainFlags

Type: **UINT**

A combination of [DXGI_SWAP_CHAIN_FLAG](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for swap-chain behavior.

[in] pCreationNodeMask

Type: **const UINT***

An array of **UINT**s, of total size *BufferCount*, where the value indicates which node the back buffer should be created on. Buffers created using [ResizeBuffers1](#) with a non-null *pCreationNodeMask* array are visible to all nodes.

[in] ppPresentQueue

Type: **IUnknown***

An array of command queues ([ID3D12CommandQueue](#) instances), of total size *BufferCount*. Each queue provided must match the corresponding creation node mask specified in the *pCreationNodeMask* array. When [Present\(\)](#) is called, in addition to rotating to the next buffer for the next frame, the swapchain will also rotate through these command queues. This allows the app to control which queue requires synchronization for a given present operation.

Return value

Type: [HRESULT](#)

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

This method is only valid to call when the swapchain was created using a D3D12 command queue ([ID3D12CommandQueue](#)) as an input device.

When a swapchain is created on a multi-GPU adapter, the backbuffers are all created on node 1 and only a single command queue is supported. **ResizeBuffers1** enables applications to create backbuffers on different nodes, allowing a different command queue to be used with each node. These capabilities enable Alternate Frame Rendering (AFR) techniques to be used with the swapchain. See [Multi-adapter systems](#).

Also see the Remarks section in [IDXGISwapChain::ResizeBuffers](#), all of which is relevant to **ResizeBuffers1**.

Requirements

Target Platform	Windows
Header	dxgi1_4.h
Library	Dxgi.lib

See also

[IDXGISwapChain3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain3::SetColorSpace1 method (dxgi1_4.h)

Article 07/27/2022

Sets the color space used by the swap chain.

Syntax

C++

```
HRESULT SetColorSpace1(  
    [in] DXGI_COLOR_SPACE_TYPE ColorSpace  
);
```

Parameters

[in] ColorSpace

Type: [DXGI_COLOR_SPACE_TYPE](#)

A [DXGI_COLOR_SPACE_TYPE](#)-typed value that specifies the color space to set.

Return value

Type: [HRESULT](#)

This method returns [S_OK](#) on success, or it returns one of the error codes that are described in the [DXGI_ERROR](#) topic.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_4.h

Library

Dxgi.lib

See also

[IDXGISwapChain3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgi1_5.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi1_5.h contains the following programming interfaces:

Interfaces

[IDXGIDevice4](#)

This interface provides updated methods to offer and reclaim resources.

[IDXGIFactory5](#)

This interface enables a single method to support variable refresh rate displays.

[IDXGIOutput5](#)

Represents an adapter output (such as a monitor). The IDXGIOutput5 interface exposes a single method to specify a list of supported formats for fullscreen surfaces.

[IDXGISwapChain4](#)

This interface exposes a single method for setting video metadata.

Structures

[DXGI_HDR_METADATA_HDR10](#)

Describes the metadata for HDR10, used when video is compressed using High Efficiency Video Coding (HEVC).

Enumerations

[DXGI_FEATURE](#)

Specifies a range of hardware features, to be used when checking for feature support.

[DXGI_HDR_METADATA_TYPE](#)

Specifies the header metadata type.

[DXGI_OFFER_RESOURCE_FLAGS](#)

Specifies flags for the OfferResources1 method.

[DXGI_RECLAIM_RESOURCE_RESULTS](#)

Specifies result flags for the ReclaimResources1 method.

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

DXGI_FEATURE enumeration (dxgi1_5.h)

Article01/31/2022

Specifies a range of hardware features, to be used when checking for feature support.

Syntax

C++

```
typedef enum DXGI_FEATURE {
    DXGI_FEATURE_PRESENT_ALLOW_TEARING = 0
} ;
```

Constants

DXGI_FEATURE_PRESENT_ALLOW_TEARING

Value: 0

The display supports tearing, a requirement of variable refresh rate displays.

Remarks

This enum is used by the [CheckFeatureSupport](#) method.

Requirements

Header

dxgi1_5.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?  

Get help at Microsoft Q&A

DXGI_HDR_METADATA_HDR10 structure (dxgi1_5.h)

Article09/01/2022

Describes the metadata for HDR10, used when video is compressed using High Efficiency Video Coding (HEVC). This is used to describe the capabilities of the display used to master the content and the luminance values of the content.

Syntax

C++

```
typedef struct DXGI_HDR_METADATA_HDR10 {
    UINT16 RedPrimary[2];
    UINT16 GreenPrimary[2];
    UINT16 BluePrimary[2];
    UINT16 WhitePoint[2];
    UINT    MaxMasteringLuminance;
    UINT    MinMasteringLuminance;
    UINT16 MaxContentLightLevel;
    UINT16 MaxFrameAverageLightLevel;
} DXGI_HDR_METADATA_HDR10;
```

Members

RedPrimary[2]

The chromaticity coordinates of the red value in the CIE1931 color space. Index 0 contains the X coordinate and index 1 contains the Y coordinate. The values are normalized to 50,000.

GreenPrimary[2]

The chromaticity coordinates of the green value in the CIE1931 color space. Index 0 contains the X coordinate and index 1 contains the Y coordinate. The values are normalized to 50,000.

BluePrimary[2]

The chromaticity coordinates of the blue value in the CIE1931 color space. Index 0 contains the X coordinate and index 1 contains the Y coordinate. The values are normalized to 50,000.

`WhitePoint[2]`

The chromaticity coordinates of the white point in the CIE1931 color space. Index 0 contains the X coordinate and index 1 contains the Y coordinate. The values are normalized to 50,000.

`MaxMasteringLuminance`

The maximum number of nits of the display used to master the content. Values are in whole nits.

`MinMasteringLuminance`

The minimum number of nits of the display used to master the content. Values are 1/10000th of a nit (0.0001 nit).

`MaxContentLightLevel`

The maximum content light level (MaxCLL). This is the nit value corresponding to the brightest pixel used anywhere in the content.

`MaxFrameAverageLightLevel`

The maximum frame average light level (MaxFALL). This is the nit value corresponding to the average luminance of the frame which has the brightest average luminance anywhere in the content.

Remarks

This structure represents the definition of HDR10 metadata used with HEVC, not HDR10 metadata for ST.2086. These are closely related but defined differently.

Example: Mastering display with DCI-P3 color primaries and D65 white point, maximum luminance of 1000 nits and minimum luminance of 0.001 nits; content has maximum luminance of 2000 nits and maximum frame average light level (MaxFALL) of 500 nits.

C++

```
RedPrimary[0] = 0.680 * 50000;
RedPrimary[1] = 0.320 * 50000;
GreenPrimary[0] = 0.265 * 50000;
GreenPrimary[1] = 0.690 * 50000;
BluePrimary[0] = 0.150 * 50000;
BluePrimary[1] = 0.060 * 50000;
WhitePoint[0] = 0.3127 * 50000;
WhitePoint[1] = 0.3290 * 50000;
```

```
MaxMasteringLuminance = 1000;  
MinMasteringLuminance = 0.001 * 10000;  
MaxContentLightLevel = 2000;  
MaxFrameAverageLightLevel = 500;
```

This structure is used in conjunction with the [SetHDRMetaData](#) method.

Requirements

Header	dxgi1_5.h
--------	-----------

See also

[DXGI 1.5 Improvements](#)

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_HDR_METADATA_TYPE enumeration (dxgi1_5.h)

Article 01/31/2022

Specifies the header metadata type.

Syntax

C++

```
typedef enum DXGI_HDR_METADATA_TYPE {
    DXGI_HDR_METADATA_TYPE_NONE = 0,
    DXGI_HDR_METADATA_TYPE_HDR10 = 1,
    DXGI_HDR_METADATA_TYPE_HDR10PLUS = 2
};
```

Constants

`DXGI_HDR_METADATA_TYPE_NONE`

Value: 0

Indicates there is no header metadata.

`DXGI_HDR_METADATA_TYPE_HDR10`

Value: 1

Indicates the header metadata is held by a [DXGI_HDR_METADATA_HDR10](#) structure.

`DXGI_HDR_METADATA_TYPE_HDR10PLUS`

Value: 2

Remarks

This enum is used by the [SetHDRMetaData](#) method.

Requirements

Header

dxgi1_5.h

See also

[DXGI 1.5 Improvements](#)

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_Offer_Resource_Flags enumeration (dxgi1_5.h)

Article01/31/2022

Specifies flags for the [OfferResources1](#) method.

Syntax

C++

```
typedef enum _DXGI_Offer_Resource_Flags {
    DXGI_Offer_Resource_Flag_Allow_DeCommit = 0x1
} DXGI_Offer_Resource_Flags;
```

Constants

`DXGI_Offer_Resource_Flag_Allow_DeCommit`

Value: *0x1*

Indicates the ability to allow memory de-commit by the DirectX Graphics Kernel.

Requirements

Header

dxgi1_5.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_RECLAIM_RESOURCE_RESULTS enumeration (dxgi1_5.h)

Article01/31/2022

Specifies result flags for the [ReclaimResources1](#) method.

Syntax

C++

```
typedef enum _DXGI_RECLAIM_RESOURCE_RESULTS {
    DXGI_RECLAIM_RESOURCE_RESULT_OK = 0,
    DXGI_RECLAIM_RESOURCE_RESULT_DISCARDED = 1,
    DXGI_RECLAIM_RESOURCE_RESULT_NOT_COMMITTED = 2
} DXGI_RECLAIM_RESOURCE_RESULTS;
```

Constants

`DXGI_RECLAIM_RESOURCE_RESULT_OK`

Value: 0

The surface was successfully reclaimed and has valid content. This result is identical to the *false* value returned by the older [ReclaimResources](#) API.

`DXGI_RECLAIM_RESOURCE_RESULT_DISCARDED`

Value: 1

The surface was reclaimed, but the old content was lost and must be regenerated. This result is identical to the *true* value returned by the older [ReclaimResources](#) API.

`DXGI_RECLAIM_RESOURCE_RESULT_NOT_COMMITTED`

Value: 2

Both the surface and its contents are lost and invalid. The surface must be recreated and the content regenerated in order to be used. All future use of that resource is invalid. Attempts to bind it to the pipeline or map a resource which returns this value will never succeed, and the resource cannot be reclaimed again.

Requirements

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDevice4 interface (dxgi1_5.h)

Article 07/22/2021

This interface provides updated methods to offer and reclaim resources.

Inheritance

The **IDXGIDevice4** interface inherits from [IDXGIDevice3](#). **IDXGIDevice4** also has these types of members:

Methods

The **IDXGIDevice4** interface has these methods.

[IDXGIDevice4::OfferResources1](#)

Allows the operating system to free the video memory of resources, including both discarding the content and de-committing the memory.

[IDXGIDevice4::ReclaimResources1](#)

Restores access to resources that were previously offered by calling [IDXGIDevice4::OfferResources1](#).

Remarks

The Direct3D create device functions return a Direct3D device object. This Direct3D device object implements the [IUnknown](#) interface. You can query this Direct3D device object for the device's corresponding **IDXGIDevice4** interface. To retrieve the **IDXGIDevice4** interface of a Direct3D device, use the following code:

C++

```
IDXGIDevice4 * pDXGIDevice;  
hr = g_pd3dDevice->QueryInterface(__uuidof(IDXGIDevice4), (void  
**) &pDXGIDevice);
```

Requirements

Target Platform	Windows
Header	dxgi1_5.h

See also

[DXGI Interfaces](#)

[IDXGIDevice2](#)

[IDXGIDevice3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice4::OfferResources1 method (dxgi1_5.h)

Article 10/13/2021

Allows the operating system to free the video memory of resources, including both discarding the content and de-committing the memory.

Syntax

C++

```
HRESULT OfferResources1(
    [in] UINT NumResources,
    [in] IDXGIResource* ppResources,
    [in] DXGI_OFFER_RESOURCE_PRIORITY Priority,
    [in] UINT Flags
);
```

Parameters

[in] NumResources

Type: **UINT**

The number of resources in the *ppResources* argument array.

[in] ppResources

Type: **IDXGIResource***

An array of pointers to **IDXGIResource** interfaces for the resources to offer.

[in] Priority

Type: **DXGI_OFFER_RESOURCE_PRIORITY**

A **DXGI_OFFER_RESOURCE_PRIORITY**-typed value that indicates how valuable data is.

[in] Flags

Type: **UINT**

Specifies the **DXGI_OFFER_RESOURCE_FLAGS**.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code, which can include E_INVALIDARG if a resource in the array, or the priority, is invalid.

Remarks

OfferResources1 (an extension of the original [IDXGIDevice2::OfferResources](#) API) enables D3D based applications to allow de-committing of an allocation's backing store to reduce system commit under low memory conditions. A de-committed allocation cannot be reused, so opting in to the new DXGI_OFFER_RESOURCE_FLAG_ALLOW_DECOMMIT flag means the new reclaim results must be properly handled. Refer to the flag descriptions in [DXGI_RECLAIM_RESOURCE_RESULTS](#) and the Example below.

OfferResources1 and [ReclaimResources1](#) may *not* be used interchangeably with **OfferResources** and [ReclaimResources](#).

The priority value that the *Priority* parameter specifies describes how valuable the caller considers the content to be. The operating system uses the priority value to discard resources in order of priority. The operating system discards a resource that is offered with low priority before it discards a resource that is offered with a higher priority.

If you call **OfferResources1** to offer a resource while the resource is bound to the pipeline, the resource is unbound. You cannot call **OfferResources1** on a resource that is mapped. After you offer a resource, the resource cannot be mapped or bound to the pipeline until you call the [ReclaimResources1](#) method to reclaim the resource. You cannot call **OfferResources1** to offer immutable resources.

To offer shared resources, call [OfferResources1](#) on only one of the sharing devices. To ensure exclusive access to the resources, you must use an [IDXGIMutex](#) object and then call **OfferResources1** only while you hold the mutex. In fact, you can't offer shared resources unless you use [IDXGIMutex](#) because offering shared resources without using [IDXGIMutex](#) isn't supported.

The user mode display driver might not immediately offer the resources that you specified in a call to **OfferResources1**. The driver can postpone offering them until the next call to [IDXGISwapChain::Present](#), [IDXGISwapChain1::Present1](#), or [ID3D11DeviceContext::Flush](#).

Examples

A UWP based application is being suspended to the background and wishes to offer its graphics resources back to the system, in case another application wants them. The application will reclaim these resources when it gets resumed. The application also realizes that the total available system commit is small on this platform, and is willing to allow its resources to be removed from the system commit. If the reclaim process fails because the system is out of memory, the application handles the error condition.

syntax

```
struct Texture
{
    UINT32 Width;
    UINT32 Height;
    UINT32 Mips;
    ID3D11Texture2D* pResource;
};

void Application::OfferInterfaceResources(ID3D11Device* pD3D11Device)
{
    CComPtr<IDXGIDevice4> pDXGIDevice;
    ThrowIfFailed(pD3D11Device->QueryInterface(&pDXGIDevice));

    for(Texture& t : m_Textures)
    {
        CComPtr<IDXGIResource> pDXGIResource;
        ThrowIfFailed(t.pResource->QueryInterface(&pDXGIResource));
        ThrowIfFailed(pDXGIDevice->OfferResources1(1, &pDXGIResource,
DXGI_OFFER_RESOURCE_PRIORITY_NORMAL,
DXGI_OFFER_RESOURCE_FLAG_ALLOW_DECOMMIT));
    }
}

void Application::ReclaimInterfaceResources (ID3D11Device* pD3D11Device)
{
    CComPtr<IDXGIDevice4> pDXGIDevice;
    ThrowIfFailed(pD3D11Device->QueryInterface(&pDXGIDevice));

    for(Texture& t : m_Textures)
    {
        CComPtr<IDXGIResource> pDXGIResource;
        ThrowIfFailed(t.pResource->QueryInterface(&pDXGIResource));

        DXGI_RECLAIM_RESOURCE_RESULTS Result;
        ThrowIfFailed(pDXGIDevice->ReclaimResources1(1,
&pDXGIResource, &Result));

        // If the surface lost its backing commitment, it must be recreated.

        if(Result == DXGI_RECLAIM_RESOURCE_RESULT_NOT_COMMITTED)
```

```

    {
        t.pResource-&gt;Release();
        t.pResource = CreateTexture(t.Width, t.Height, t.Mips);
    }

    // If the surface lost its content (either because it was discarded,
    or recreated
    // due to lost commitment), we must regenerate the content.

    if(Result != DXGI_RECLAIM_RESOURCE_RESULT_OK)
    {
        PopulateContent(t);
    }
}

```

Requirements

Target Platform	Windows
Header	dxgi1_5.h
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[DXGI_RECLAIM_RESOURCE_RESULTS](#)

[IDXGIDevice4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDevice4::ReclaimResources1 method (dxgi1_5.h)

Article 10/13/2021

Restores access to resources that were previously offered by calling [IDXGIDevice4::OfferResources1](#).

Syntax

C++

```
HRESULT ReclaimResources1(
    [in]    UINT                  NumResources,
    [in]    IDXGIResource        * ppResources,
    [out]   DXGI_RECLAIM_RESOURCE_RESULTS * pResults
);
```

Parameters

[in] NumResources

Type: **UINT**

The number of resources in the *ppResources* argument and *pResults* argument arrays.

[in] ppResources

Type: **IDXGIResource***

An array of pointers to [IDXGIResource](#) interfaces for the resources to reclaim.

[out] pResults

Type: [DXGI_RECLAIM_RESOURCE_RESULTS*](#)

A pointer to an array that receives [DXGI_RECLAIM_RESOURCE_RESULTS](#) values. Each value in the array corresponds to a resource at the same index that the *ppResources* parameter specifies. The caller can pass in **NULL**, if the caller intends to fill the resources with new content regardless of whether the old content was discarded.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code, including E_INVALIDARG if the resources are invalid.

Remarks

After you call [OfferResources1](#) to offer one or more resources, you must call [ReclaimResources1](#) before you can use those resources again.

To reclaim shared resources, call [ReclaimResources1](#) only on one of the sharing devices. To ensure exclusive access to the resources, you must use an [IDXGIKeyedMutex](#) object and then call [ReclaimResources1](#) only while you hold the mutex.

Requirements

Target Platform	Windows
Header	dxgi1_5.h
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[IDXGIDevice4](#)

[ReclaimResources](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory5 interface (dxgi1_5.h)

Article07/22/2021

This interface enables a single method to support variable refresh rate displays.

Inheritance

The **IDXGIFactory5** interface inherits from [IDXGIFactory4](#). **IDXGIFactory5** also has these types of members:

Methods

The **IDXGIFactory5** interface has these methods.

[IDXGIFactory5::CheckFeatureSupport](#)

Used to check for hardware feature support.

Requirements

Target Platform

Windows

Header

dxgi1_5.h

See also

[DXGI Interfaces](#)

[IDXGIFactory1](#)

[IDXGIFactory2](#)

[IDXGIFactory3](#)

[IDXGIFactory4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory5::CheckFeatureSupport method (dxgi1_5.h)

Article 10/13/2021

Used to check for hardware feature support.

Syntax

C++

```
HRESULT CheckFeatureSupport(
    DXGI_FEATURE Feature,
    [in, out] void* pFeatureSupportData,
    UINT FeatureSupportDataSize
);
```

Parameters

Feature

Type: [DXGI FEATURE](#)

Specifies one member of [DXGI FEATURE](#) to query support for.

[in, out] pFeatureSupportData

Type: [void*](#)

Specifies a pointer to a buffer that will be filled with data that describes the feature support.

FeatureSupportDataSize

Type: [UINT](#)

The size, in bytes, of *pFeatureSupportData*.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code.

Remarks

Refer to the description of [DXGI_SWAP_CHAIN_FLAG_ALLOW_TEARING](#).

Requirements

Target Platform	Windows
Header	dxgi1_5.h
Library	Dxgi.lib

See also

[IDXGIFactory5](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput5 interface (dxgi1_5.h)

Article07/22/2021

Represents an adapter output (such as a monitor). The **IDXGIOutput5** interface exposes a single method to specify a list of supported formats for fullscreen surfaces.

Inheritance

The **IDXGIOutput5** interface inherits from [IDXGIOutput4](#). **IDXGIOutput5** also has these types of members:

Methods

The **IDXGIOutput5** interface has these methods.

[IDXGIOutput5::DuplicateOutput1](#)

Allows specifying a list of supported formats for fullscreen surfaces that can be returned by the **IDXGIOutputDuplication** object.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_5.h

See also

[DXGI Interfaces](#)

[IDXGIOutput4](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput5::DuplicateOutput1 method (dxgi1_5.h)

Article 10/13/2021

Allows specifying a list of supported formats for fullscreen surfaces that can be returned by the [IDXGIOutputDuplication](#) object.

Syntax

C++

```
HRESULT DuplicateOutput1(
    [in] IUnknown* pDevice,
    [in] UINT Flags,
    [in] UINT SupportedFormatsCount,
    [in] const DXGI_FORMAT* pSupportedFormats,
    [out] IDXGIOutputDuplication** ppOutputDuplication
);
```

Parameters

[in] pDevice

Type: [IUnknown*](#)

A pointer to the Direct3D device interface that you can use to process the desktop image. This device must be created from the adapter to which the output is connected.

Flags

Type: [UINT](#)

Reserved for future use; must be zero.

[in] SupportedFormatsCount

Type: [UINT](#)

Specifies the number of supported formats.

[in] pSupportedFormats

Type: [const DXGI_FORMAT*](#)

Specifies an array, of length *SupportedFormatsCount* of [DXGI_FORMAT](#) entries.

[out] ppOutputDuplication

Type: [IDXGIOutputDuplication**](#)

A pointer to a variable that receives the new [IDXGIOutputDuplication](#) interface.

Return value

Type: [HRESULT](#)

- S_OK if [DuplicateOutput1](#) successfully created the desktop duplication interface.
- E_INVALIDARG for one of the following reasons:
 - The specified device (*pDevice*) is invalid, was not created on the correct adapter, or was not created from [IDXGIFactory1](#) (or a later version of a DXGI factory interface that inherits from [IDXGIFactory1](#)).
 - The calling application is already duplicating this desktop output.
- E_ACCESSDENIED if the application does not have access privilege to the current desktop image. For example, only an application that runs at LOCAL_SYSTEM can access the secure desktop.
- DXGI_ERROR_UNSUPPORTED if the created [IDXGIOutputDuplication](#) interface does not support the current desktop mode or scenario. For example, 8bpp and non-DWM desktop modes are not supported.

If [DuplicateOutput1](#) fails with DXGI_ERROR_UNSUPPORTED, the application can wait for system notification of desktop switches and mode changes and then call [DuplicateOutput1](#) again after such a notification occurs. For more information, see the desktop switch ([EVENT_SYSTEM_DESKTOPSWITCH](#)) and mode change notification ([WM_DISPLAYCHANGE](#)).

- DXGI_ERROR_NOT_CURRENTLY_AVAILABLE if DXGI reached the limit on the maximum number of concurrent duplication applications (default of four). Therefore, the calling application cannot create any desktop duplication interfaces until the other applications close.
- DXGI_ERROR_SESSION_DISCONNECTED if [DuplicateOutput1](#) failed because the session is currently disconnected.
- Other error codes are described in the [DXGI_ERROR](#) topic.

Remarks

This method allows directly receiving the original back buffer format used by a running fullscreen application. For comparison, using the original [DuplicateOutput](#) function

always converts the fullscreen surface to a 32-bit BGRA format. In cases where the current fullscreen application is using a different buffer format, a conversion to 32-bit BGRA incurs a performance penalty. Besides the performance benefit of being able to skip format conversion, using [DuplicateOutput1](#) also allows receiving the full gamut of colors in cases where a high-color format (such as R10G10B10A2) is being presented.

The *pSupportedFormats* array should only contain display scan-out formats. See [Format Support for Direct3D Feature Level 11.0 Hardware](#) for required scan-out formats at each feature level. If the current fullscreen buffer format is not contained in the *pSupportedFormats* array, DXGI will pick one of the supplied formats and convert the fullscreen buffer to that format before returning from [IDXGIOutputDuplication::AcquireNextFrame](#). The list of supported formats should always contain DXGI_FORMAT_B8G8R8A8_UNORM, as this is the most common format for the desktop.

Requirements

Target Platform	Windows
Header	dxgi1_5.h
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[DuplicateOutput](#)

[IDXGIOutput5](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGISwapChain4 interface (dxgi1_5.h)

Article 07/22/2021

This interface exposes a single method for setting video metadata.

Inheritance

The **IDXGISwapChain4** interface inherits from [IDXGISwapChain3](#). **IDXGISwapChain4** also has these types of members:

Methods

The **IDXGISwapChain4** interface has these methods.

[IDXGISwapChain4::SetHDRMetaData](#)

This method sets High Dynamic Range (HDR) and Wide Color Gamut (WCG) header metadata.

Requirements

Target Platform

Windows

Header

dxgi1_5.h

See also

[DXGI Interfaces](#)

[IDXGISwapChain3](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGISwapChain4::SetHDRMetaData method (dxgi1_5.h)

Article 11/03/2022

⚠️ Warning

It is no longer recommended for apps to explicitly set HDR metadata on their swap chain using `SetHDRMetaData`. Windows does not guarantee that swap chain metadata is sent to the monitor, and monitors do not handle HDR metadata consistently. Therefore it's recommended that apps always tone-map content into the range reported by the monitor. For more details on how to write apps that react dynamically to monitor capabilities, see [Using DirectX with high dynamic range displays and Advanced Color](#).

See Remarks for more details.

This method sets High Dynamic Range (HDR) and Wide Color Gamut (WCG) header metadata.

Syntax

C++

```
HRESULT SetHDRMetaData(  
    [in]          DXGI_HDR_METADATA_TYPE Type,  
    [in]          UINT             Size,  
    [in, optional] void            *pMetaData  
) ;
```

Parameters

[in] Type

Type: [DXGI_HDR_METADATA_TYPE](#)

Specifies one member of the [DXGI_HDR_METADATA_TYPE](#) enum.

[in] Size

Type: [UINT](#)

Specifies the size of *pMetaData*, in bytes.

[in, optional] *pMetaData*

Type: **void***

Specifies a void pointer that references the metadata, if it exists. Refer to the [DXGI_HDR_METADATA_HDR10](#) structure.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code.

Remarks

This method sets metadata to enable a monitor's output to be adjusted depending on its capabilities. However it does not change how pixel values are interpreted by Windows or monitors. To adjust the color space of the swap chain, use [SetColorSpace1](#) instead.

Applications should not rely on the metadata being sent to the monitor as the the metadata may be ignored. Monitors do not consistently process HDR metadata, resulting in varied appearance of your content across different monitors. In order to ensure more consistent output across a range of monitors, devices, and use cases, it is recommended to not use [SetHDRMetaData](#) and to instead tone-map content into the gamut and luminance range supported by the monitor. See [IDXGIOutput6::GetDesc1](#) to retrieve the monitor's supported gamut and luminance range. Monitors adhering to the VESA DisplayHDR standard will automatically perform a form of clipping for content outside of the monitor's supported gamut and luminance range.

For more details on how to write apps that react dynamically to monitor capabilities, see [Using DirectX with high dynamic range displays and Advanced Color](#).

Requirements

Target Platform	Windows
Header	dxgi1_5.h

Library	Dxgi1_5.lib
DLL	Dxgi1_5.dll

See also

[DXGI 1.5 Improvements](#)

[IDXGISwapChain4](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

dxgi1_6.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgi1_6.h contains the following programming interfaces:

Interfaces

[IDXGIAAdapter4](#)

This interface represents a display subsystem, and extends this family of interfaces to expose a method to check for an adapter's compatibility with Arbitrary Code Guard (ACG).

[IDXGIFactory6](#)

This interface enables a single method that enumerates graphics adapters based on a given GPU preference.

[IDXGIFactory7](#)

This interface enables registration for notifications to detect adapter enumeration state changes.

[IDXGIOOutput6](#)

Represents an adapter output (such as a monitor). The IDXGIOOutput6 interface exposes methods to provide specific monitor capabilities.

Functions

[DXGIDeclareAdapterRemovalSupport](#)

Allows a process to indicate that it's resilient to any of its graphics devices being removed.

[DXGIDisableVBlankVirtualization](#)

Disables v-blank virtualization for the process. This virtualization is used by the dynamic refresh rate (DRR) feature by default for all swap chains to maintain a steady virtualized present rate and v-blank cadence from [IDXGIOutput::WaitForVBlank](#). By disabling virtualization, these APIs will see the changing refresh rate.

Structures

[DXGI_ADAPTER_DESC3](#)

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.6.

[DXGI_OUTPUT_DESC1](#)

Describes an output or physical connection between the adapter (video card) and a device, including additional information about color capabilities and connection type.

Enumerations

[DXGI_ADAPTER_FLAG3](#)

Identifies the type of DXGI adapter. (DXGI_ADAPTER_FLAG3)

[DXGI_GPU_PREFERENCE](#)

The preference of GPU for the app to run on.

[DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAGS](#)

Describes which levels of hardware composition are supported.

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_ADAPTER_DESC3 structure (dxgi1_6.h)

Article09/01/2022

Describes an adapter (or video card) that uses Microsoft DirectX Graphics Infrastructure (DXGI) 1.6.

Syntax

C++

```
typedef struct DXGI_ADAPTER_DESC3 {
    WCHAR Description[128];
    UINT VendorId;
    UINT DeviceId;
    UINT SubSysId;
    UINT Revision;
    SIZE_T DedicatedVideoMemory;
    SIZE_T DedicatedSystemMemory;
    SIZE_T SharedSystemMemory;
    LUID AdapterLuid;
    DXGI_ADAPTER_FLAG3 Flags;
    DXGI_GRAPHICS_PREEMPTION_GRANULARITY GraphicsPreemptionGranularity;
    DXGI_COMPUTE_PREEMPTION_GRANULARITY ComputePreemptionGranularity;
} DXGI_ADAPTER_DESC3;
```

Members

Description[128]

A string that contains the adapter description.

VendorId

The PCI ID of the hardware vendor.

DeviceId

The PCI ID of the hardware device.

SubSysId

The PCI ID of the sub system.

Revision

The PCI ID of the revision number of the adapter.

DedicatedVideoMemory

The number of bytes of dedicated video memory that are not shared with the CPU.

DedicatedSystemMemory

The number of bytes of dedicated system memory that are not shared with the CPU. This memory is allocated from available system memory at boot time.

SharedSystemMemory

The number of bytes of shared system memory. This is the maximum value of system memory that may be consumed by the adapter during operation. Any incidental memory consumed by the driver as it manages and uses video memory is additional.

AdapterLuid

A unique value that identifies the adapter. See [LUID](#) for a definition of the structure. LUID is defined in dxgi.h.

Flags

A value of the [DXGI_ADAPTER_FLAG3](#) enumeration that describes the adapter type. The [DXGI_ADAPTER_FLAG_REMOTE](#) flag is reserved.

GraphicsPreemptionGranularity

A value of the [DXGI_GRAPHICS_PREEMPTION_GRANULARITY](#) enumerated type that describes the granularity level at which the GPU can be preempted from performing its current graphics rendering task.

ComputePreemptionGranularity

A value of the [DXGI_COMPUTE_PREEMPTION_GRANULARITY](#) enumerated type that describes the granularity level at which the GPU can be preempted from performing its current compute task.

Remarks

The [DXGI_ADAPTER_DESC3](#) structure provides a DXGI 1.6 description of an adapter. This structure is initialized by using the [IDXGIAdapter4::GetDesc3](#) method.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Header	dxgi1_6.h

See also

[DXGI Structures](#)

[IDXGIAAdapter4::GetDesc3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_ADAPTER_FLAG3 enumeration (dxgi1_6.h)

Article07/27/2022

Identifies the type of DXGI adapter.

Syntax

C++

```
typedef enum DXGI_ADAPTER_FLAG3 {
    DXGI_ADAPTER_FLAG3_NONE = 0,
    DXGI_ADAPTER_FLAG3_REMOTE = 1,
    DXGI_ADAPTER_FLAG3_SOFTWARE = 2,
    DXGI_ADAPTER_FLAG3_ACG_COMPATIBLE = 4,
    DXGI_ADAPTER_FLAG3_SUPPORT_MONITORED_FENCES = 8,
    DXGI_ADAPTER_FLAG3_SUPPORT_NON_MONITORED_FENCES = 0x10,
    DXGI_ADAPTER_FLAG3_KEYED_MUTEX_CONFORMANCE = 0x20,
    DXGI_ADAPTER_FLAG3_FORCE_DWORD = 0xffffffff
} ;
```

Constants

`DXGI_ADAPTER_FLAG3_NONE`

Value: 0

Specifies no flags.

`DXGI_ADAPTER_FLAG3_REMOTE`

Value: 1

Value always set to 0. This flag is reserved.

`DXGI_ADAPTER_FLAG3_SOFTWARE`

Value: 2

Specifies a software adapter. For more info about this flag, see [new info in Windows 8 about enumerating adapters](#).

Direct3D 11: This enumeration value is supported starting with Windows 8.

`DXGI_ADAPTER_FLAG3_ACG_COMPATIBLE`

Value: 4

Specifies that the adapter's driver has been confirmed to work in an OS process where Arbitrary Code Guard (ACG) is enabled (i.e. dynamic code generation is disallowed).

`DXGI_ADAPTER_FLAG3_SUPPORT_MONITORED_FENCES`

Value: *8*

Specifies that the adapter supports monitored fences. These adapters support the [ID3D12Device::CreateFence](#) and [ID3D11Device5::CreateFence](#) functions.

`DXGI_ADAPTER_FLAG3_SUPPORT_NON_MONITORED_FENCES`

Value: *0x10*

Specifies that the adapter supports non-monitored fences. These adapters support the [ID3D12Device::CreateFence](#) function together with the [D3D12_FENCE_FLAG_NON_MONITORED](#) flag.

Note For adapters that support both monitored and non-monitored fences, non-monitored fences are only supported when created with the [D3D12_FENCE_FLAG_SHARED](#) and [D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER](#) flags. Monitored fences should always be used by supporting adapters unless communicating with an adapter that only supports non-monitored fences.

`DXGI_ADAPTER_FLAG3_KEYED_MUTEX_CONFORMANCE`

Value: *0x20*

Specifies that the adapter claims keyed mutex conformance. This signals a stronger guarantee that the [IDXGIKeyedMutex](#) interface behaves correctly.

`DXGI_ADAPTER_FLAG3_FORCE_DWORD`

Value: *0xffffffff*

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

Remarks

The `DXGI_ADAPTER_FLAG3` enumerated type is used by the `Flags` member of the [DXGI_ADAPTER_DESC3](#) structure to identify the type of DXGI adapter.

Requirements

Header

dxgi1_6.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_GPU_PREFERENCE enumeration (dxgi1_6.h)

Article 01/31/2022

The preference of GPU for the app to run on.

Syntax

C++

```
typedef enum DXGI_GPU_PREFERENCE {
    DXGI_GPU_PREFERENCE_UNSPECIFIED = 0,
    DXGI_GPU_PREFERENCE_MINIMUM_POWER,
    DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE
} ;
```

Constants

`DXGI_GPU_PREFERENCE_UNSPECIFIED`

Value: 0

No preference of GPU.

`DXGI_GPU_PREFERENCE_MINIMUM_POWER`

Preference for the minimum-powered GPU (such as an integrated graphics processor, or iGPU).

`DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE`

Preference for the highest performing GPU, such as a discrete graphics processor (dGPU) or external graphics processor (xGPU).

Remarks

This enumeration is used in the [IDXGIFactory6::EnumAdapterByGpuPreference](#) method.

Requirements

Header

dxgi1_6.h

See also

[DXGI Enumerations](#)

[xGPU UWP sample ↗](#)

[xGPU desktop sample ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_HARDWARE_COMPOSITION_SUPP ORT_FLAGS enumeration (dxgi1_6.h)

Article 01/31/2022

Describes which levels of hardware composition are supported.

Syntax

C++

```
typedef enum DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAGS {
    DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_FULLSCREEN = 1,
    DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_WINDOWED = 2,
    DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_CURSOR_STRETCHED = 4
} ;
```

Constants

`DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_FULLSCREEN`

Value: 1

This flag specifies that swapchain composition can be facilitated in a performant manner using hardware for fullscreen applications.

`DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_WINDOWED`

Value: 2

This flag specifies that swapchain composition can be facilitated in a performant manner using hardware for windowed applications.

`DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAG_CURSOR_STRETCHED`

Value: 4

This flag specifies that swapchain composition facilitated using hardware can cause the cursor to appear stretched.

Remarks

Values of this enumeration are returned from the

[IDXGIOutput6::CheckHardwareCompositionSupport](#) method in the *pFlags* out parameter.

Requirements

Header	dxgi1_6.h
--------	-----------

See also

[DXGI Enumerations](#)

[IDXGIOutput6::CheckHardwareCompositionSupport method](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_OUTPUT_DESC1 structure (dxgi1_6.h)

Article09/01/2022

Describes an output or physical connection between the adapter (video card) and a device, including additional information about color capabilities and connection type.

Syntax

C++

```
typedef struct DXGI_OUTPUT_DESC1 {
    WCHAR             DeviceName[32];
    RECT              DesktopCoordinates;
    BOOL              AttachedToDesktop;
    DXGI_MODE_ROTATION Rotation;
    HMONITOR          Monitor;
    UINT              BitsPerColor;
    DXGI_COLOR_SPACE_TYPE ColorSpace;
    FLOAT             RedPrimary[2];
    FLOAT             GreenPrimary[2];
    FLOAT             BluePrimary[2];
    FLOAT             WhitePoint[2];
    FLOAT             MinLuminance;
    FLOAT             MaxLuminance;
    FLOAT             MaxFullFrameLuminance;
} DXGI_OUTPUT_DESC1;
```

Members

DeviceName[32]

Type: **WCHAR[32]**

A string that contains the name of the output device.

DesktopCoordinates

Type: **RECT**

A **RECT** structure containing the bounds of the output in desktop coordinates. Desktop coordinates depend on the dots per inch (DPI) of the desktop. For info about writing DPI-aware Win32 apps, see [High DPI](#).

`AttachedToDesktop`

Type: **BOOL**

True if the output is attached to the desktop; otherwise, false.

`Rotation`

Type: **DXGI_MODE_ROTATION**

A member of the [DXGI_MODE_ROTATION](#) enumerated type describing on how an image is rotated by the output.

`Monitor`

Type: **HMONITOR**

An [HMONITOR](#) handle that represents the display monitor. For more information, see [HMONITOR and the Device Context](#).

`BitsPerColor`

Type: **UINT**

The number of bits per color channel for the active wire format of the display attached to this output.

`ColorSpace`

Type: **DXGI_COLOR_SPACE_TYPE**

The current advanced color capabilities of the display attached to this output. Specifically, whether its capable of reproducing color and luminance values outside of the sRGB color space. A value of `DXGI_COLOR_SPACE_RGB_FULL_G22_NONE_P709` indicates that the display is limited to SDR/sRGB; A value of `DXGI_COLOR_SPACE_RGB_FULL_G2048_NONE_P2020` indicates that the display supports advanced color capabilities.

For detailed luminance and color capabilities, see additional members of this struct.

`RedPrimary[2]`

Type: **FLOAT[2]**

The red color primary, in xy coordinates, of the display attached to this output. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`GreenPrimary[2]`

Type: `FLOAT[2]`

The green color primary, in xy coordinates, of the display attached to this output. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`BluePrimary[2]`

Type: `FLOAT[2]`

The blue color primary, in xy coordinates, of the display attached to this output. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`WhitePoint[2]`

Type: `FLOAT[2]`

The white point, in xy coordinates, of the display attached to this output. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`MinLuminance`

Type: `FLOAT`

The minimum luminance, in nits, that the display attached to this output is capable of rendering. Content should not exceed this minimum value for optimal rendering. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`MaxLuminance`

Type: `FLOAT`

The maximum luminance, in nits, that the display attached to this output is capable of rendering; this value is likely only valid for a small area of the panel. Content should not exceed this minimum value for optimal rendering. This value will usually come from the EDID of the corresponding display or sometimes from an override.

`MaxFullFrameLuminance`

Type: `FLOAT`

The maximum luminance, in nits, that the display attached to this output is capable of rendering; unlike MaxLuminance, this value is valid for a color that fills the entire area of the panel. Content should not exceed this value across the entire panel for optimal rendering. This value will usually come from the EDID of the corresponding display or sometimes from an override.

Remarks

The `DXGI_OUTPUT_DESC1` structure is initialized by the [IDXGIOutput6::GetDesc1](#) method.

Requirements

Header	dxgi1_6.h
--------	-----------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGIDeclareAdapterRemovalSupport

function (dxgi1_6.h)

Article 08/03/2021

Allows a process to indicate that it's resilient to any of its graphics devices being removed.

Syntax

C++

```
HRESULT DXGIDeclareAdapterRemovalSupport();
```

Return value

Type: **HRESULT**

Returns **S_OK** if successful; an error code otherwise. If this function is called after device creation, it returns **DXGI_ERROR_INVALID_CALL**. If this is not the first time that this function is called, it returns **DXGI_ERROR_ALREADY_EXISTS**. For a full list of error codes, see [DXGI_ERROR](#).

Remarks

This function is graphics API-agnostic, meaning that apps running on other APIs, such as OpenGL and Vulkan, would also apply.

This function should be called once per process and before any device creation.

Requirements

Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	None supported
Target Platform	Windows

Header	dxgi1_6.h
Library	Dxgi.lib
DLL	Dxgi.dll

See also

[DXGI AdapterRemovalSupport test sample ↗](#)

[DXGI Functions](#)

[xGPU UWP sample ↗](#)

[xGPU desktop sample ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGIDisableVBlankVirtualization function (dxgi1_6.h)

Article 05/24/2022

Disables v-blank virtualization for the process. This virtualization is used by the dynamic refresh rate (DRR) feature by default for all swap chains to maintain a steady virtualized present rate and v-blank cadence from [IDXGIOutput::WaitForVBlank](#). By disabling virtualization, these APIs will see the changing refresh rate.

Syntax

C++

```
HRESULT DXGIDisableVBlankVirtualization();
```

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; an error code otherwise. For a full list of error codes, see [DXGI_ERROR](#).

Remarks

By default, a DXGI swap chain is unable to observe the changing refresh rate that's caused by the dynamic refresh rate (DRR) feature (see the blog post [Dynamic refresh rate—Get the best of both worlds](#)). Instead, a swap chain is virtualized to always see a fraction of the refresh rate—60Hz if the DRR mode is 120Hz.

DXGIDisableVBlankVirtualization disables that virtualization for the entire process. Your application will then see v-blank timings change as the system boosts between 60Hz and 120Hz, and frames will arrive at the corresponding times for each rate, with present statistics reflecting those changes.

You should call **DXGIDisableVBlankVirtualization** once per process, before creating any swap chains or calling [IDXGIOutput::WaitForVBlank](#). It can't be disabled for the lifetime of the process, so any changes in v-blank timing or statistics from DRR boosting will remain observable to the process.

You can find more information on how Dynamic Refresh Rate works in the [Compositor clock](#) topic.

Requirements

Minimum supported client	Windows 11, version 22502 [desktop apps only]
Minimum supported server	Windows 11, version 22502 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h
Library	dxgi.lib
DLL	dxgi.dll

See also

- [Dynamic refresh rate—Get the best of both worlds ↗](#)
- [Compositor clock](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter4 interface (dxgi1_6.h)

Article07/22/2021

This interface represents a display subsystem, and extends this family of interfaces to expose a method to check for an adapter's compatibility with Arbitrary Code Guard (ACG).

Inheritance

The [IDXGIAdapter4](#) interface inherits from [IDXGIAdapter3](#). [IDXGIAdapter4](#) also has these types of members:

Methods

The [IDXGIAdapter4](#) interface has these methods.

[IDXGIAdapter4::GetDesc3](#)

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.6 description of an adapter or video card. This description includes information about ACG compatibility.

Remarks

For more details, refer to the [Residency](#) section of the D3D12 documentation.

Requirements

Target Platform	Windows
Header	dxgi1_6.h

See also

[DXGI Interfaces](#)

[IDXGIAdapter3](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIAdapter4::GetDesc3 method (dxgi1_6.h)

Article 10/13/2021

Gets a Microsoft DirectX Graphics Infrastructure (DXGI) 1.6 description of an adapter or video card. This description includes information about ACG compatibility.

Syntax

C++

```
HRESULT GetDesc3(  
    [out] DXGI_ADAPTER_DESC3 *pDesc  
) ;
```

Parameters

[out] pDesc

A pointer to a [DXGI_ADAPTER_DESC3](#) structure that describes the adapter.

This parameter must not be **NULL**. On **feature level** 9 graphics hardware, early versions of **GetDesc3** ([GetDesc1](#), and [GetDesc](#)) return zeros for the PCI ID in the **VendorId**, **DeviceId**, **SubSysId**, and **Revision** members of the adapter description structure and "Software Adapter" for the description string in the **Description** member. **GetDesc3** and [GetDesc2](#) return the actual feature level 9 hardware values in these members.

Return value

Returns **S_OK** if successful; otherwise, returns **E_INVALIDARG** if the *pDesc* parameter is **NULL**.

Remarks

Use the **GetDesc3** method to get a DXGI 1.6 description of an adapter. To get a DXGI 1.2 description, use the [IDXGIAdapter2::GetDesc2](#) method. To get a DXGI 1.1 description, use the [IDXGIAdapter1::GetDesc1](#) method. To get a DXGI 1.0 description, use the [IDXGIAdapter::GetDesc](#) method.

The Windows Display Driver Model (WDDM) scheduler can preempt the graphics processing unit (GPU)'s execution of application tasks. The granularity at which the GPU can be preempted from performing its current task in the WDDM 1.1 or earlier driver model is a direct memory access (DMA) buffer for graphics tasks or a compute packet for compute tasks. The GPU can switch between tasks only after it completes the currently executing unit of work, a DMA buffer or a compute packet.

A DMA buffer is the largest independent unit of graphics work that the WDDM scheduler can submit to the GPU. This buffer contains a set of GPU instructions that the WDDM driver and GPU use. A compute packet is the largest independent unit of compute work that the WDDM scheduler can submit to the GPU. A compute packet contains dispatches (for example, calls to the [ID3D11DeviceContext::Dispatch](#) method), which contain thread groups. The WDDM 1.2 or later driver model allows the GPU to be preempted at finer granularity levels than a DMA buffer or compute packet. You can use the [GetDesc3](#) or [GetDesc2](#) methods to retrieve the granularity levels for graphics and compute tasks.

Requirements

Minimum supported client	Windows 8 and Platform Update for Windows 7 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 and Platform Update for Windows Server 2008 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgi1_6.h
Library	DXGI.lib
DLL	Dxgi.dll

See also

[IDXGIAAdapter4](#)

Feedback



Was this page helpful? Yes No

Get help at Microsoft Q&A

IDXGIFactory6 interface (dxgi1_6.h)

Article07/22/2021

This interface enables a single method that enumerates graphics adapters based on a given GPU preference.

Inheritance

The **IDXGIFactory6** interface inherits from [IDXGIFactory5](#). **IDXGIFactory6** also has these types of members:

Methods

The **IDXGIFactory6** interface has these methods.

[IDXGIFactory6::EnumAdapterByGpuPreference](#)

Enumerates graphics adapters based on a given GPU preference.

Requirements

Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server, version 1709 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h

See also

[DXGI Interfaces](#)

[IDXGIFactory](#)

[IDXGIFactory1](#)

[IDXGIFactory2](#)

[IDXGIFactory3](#)

[IDXGIFactory4](#)

[IDXGIFactory5](#)

[xGPU UWP sample ↗](#)

[xGPU desktop sample ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory6::EnumAdapterByGpuPreference method (dxgi1_6.h)

Article 10/13/2021

Enumerates graphics adapters based on a given GPU preference.

Syntax

C++

```
HRESULT EnumAdapterByGpuPreference(
    [in]     UINT             Adapter,
    [in]     DXGI_GPU_PREFERENCE GpuPreference,
    [in]     REFIID            riid,
    [out]    void              **ppvAdapter
);
```

Parameters

[in] Adapter

Type: **UINT**

The index of the adapter to enumerate. The indices are in order of the preference specified in *GpuPreference*—for example, if **DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE** is specified, then the highest-performing adapter is at index 0, the second-highest is at index 1, and so on.

[in] GpuPreference

Type: **DXGI_GPU_PREFERENCE**

The GPU preference for the app.

[in] riid

Type: **REFIID**

The globally unique identifier (GUID) of the **IDXGIAdapter** object referenced by the *ppvAdapter* parameter.

[out] ppvAdapter

Type: **void****

The address of an [IDXGIAdapter](#) interface pointer to the adapter.

This parameter must not be NULL.

Return value

Type: **HRESULT**

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

This method allows developers to select which GPU they think is most appropriate for each device their app creates and utilizes.

This method is similar to [IDXGIFactory1::EnumAdapters1](#), but it accepts a GPU preference to reorder the adapter enumeration. It returns the appropriate [IDXGIAdapter](#) for the given GPU preference. It is meant to be used in conjunction with the **D3DCreateDevice** functions, which take in an [IDXGIAdapter](#).

When **DXGI_GPU_PREFERENCE_UNSPECIFIED** is specified for the *GpuPreference* parameter, this method is equivalent to calling [IDXGIFactory1::EnumAdapters1](#).

When **DXGI_GPU_PREFERENCE_MINIMUM_POWER** is specified for the *GpuPreference* parameter, the order of preference for the adapter returned in *ppvAdapter* will be:

1. iGPUs (integrated GPUs)
2. dGPUs (discrete GPUs)
3. xGPUs (external GPUs)

When **DXGI_GPU_PREFERENCE_HIGH_PERFORMANCE** is specified for the *GpuPreference* parameter, the order of preference for the adapter returned in *ppvAdapter* will be:

1. xGPUs
2. dGPUs
3. iGPUs

Requirements

Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server, version 1709 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h
Library	Dxgi.lib

See also

[IDXGIFactory6](#)

[xGPU UWP sample](#) ↗

[xGPU desktop sample](#) ↗

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIFactory7 interface (dxgi1_6.h)

Article07/22/2021

This interface enables registration for notifications to detect adapter enumeration state changes.

Inheritance

The **IDXGIFactory7** interface inherits from [IDXGIFactory6](#). **IDXGIFactory7** also has these types of members:

Methods

The **IDXGIFactory7** interface has these methods.

IDXGIFactory7::RegisterAdaptersChangedEvent
Registers to receive notification of changes whenever the adapter enumeration state changes.
IDXGIFactory7::UnregisterAdaptersChangedEvent
Unregisters an event to stop receiving notifications when the adapter enumeration state changes.

Requirements

Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server, version 1709 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

IDXGIFactory7::RegisterAdaptersChange dEvent method (dxgi1_6.h)

Article 10/13/2021

Registers to receive notification of changes whenever the adapter enumeration state changes.

Syntax

C++

```
HRESULT RegisterAdaptersChangedEvent(
    [in]      HANDLE hEvent,
    [in, out] DWORD *pdwCookie
);
```

Parameters

[in] hEvent

A handle to the event object.

[in, out] pdwCookie

A key value for the registered event.

Return value

Returns S_OK if successful; an error code otherwise.

Requirements

Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server, version 1709 [desktop apps only]
Target Platform	Windows

Header	dxgi1_6.h
Library	Dxgi.lib

See also

[IDXGIFactory7](#)

[UnregisterAdaptersChangedEvent](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIFactory7::UnregisterAdaptersChangedEvent method (dxgi1_6.h)

Article 10/13/2021

Unregisters an event to stop receiving notifications when the adapter enumeration state changes.

Syntax

C++

```
HRESULT UnregisterAdaptersChangedEvent(  
    [in] DWORD dwCookie  
>;
```

Parameters

[in] dwCookie

A key value for the event to unregister.

Return value

Returns `S_OK` if successful; an error code otherwise.

Requirements

Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server, version 1709 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h
Library	Dxgi.lib

See also

[IDXGIFactory7](#)

[RegisterAdaptersChangedEvent](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIOutput6 interface (dxgi1_6.h)

Article07/22/2021

Represents an adapter output (such as a monitor). The **IDXGIOutput6** interface exposes methods to provide specific monitor capabilities.

Inheritance

The **IDXGIOutput6** interface inherits from [IDXGIOutput5](#). **IDXGIOutput6** also has these types of members:

Methods

The **IDXGIOutput6** interface has these methods.

IDXGIOutput6::CheckHardwareCompositionSupport
Notifies applications that hardware stretching is supported.
IDXGIOutput6::GetDesc1
Get an extended description of the output that includes color characteristics and connection type.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	dxgi1_6.h

See also

[DXGI Interfaces](#)

[IDXGIOutput5](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput6::CheckHardwareCompositionSupport method (dxgi1_6.h)

Article 10/13/2021

Notifies applications that hardware stretching is supported.

Syntax

C++

```
HRESULT CheckHardwareCompositionSupport(
    [out] UINT *pFlags
);
```

Parameters

[out] pFlags

Type: **UINT***

A bitfield of [DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAGS](#) enumeration values describing which types of hardware composition are supported. The values are bitwise OR'd together.

Return value

Type: [HRESULT](#)

Returns a code that indicates success or failure.

Requirements

Target Platform	Windows
Header	dxgi1_6.h
Library	DXGI.lib

See also

[DXGI Interfaces](#)

[DXGI_HARDWARE_COMPOSITION_SUPPORT_FLAGS enumeration](#)

[IDXGIOutput6 interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIOutput6::GetDesc1 method (dxgi1_6.h)

Article11/03/2022

Get an extended description of the output that includes color characteristics and connection type.

Syntax

C++

```
HRESULT GetDesc1(  
    [out] DXGI_OUTPUT_DESC1 *pDesc  
);
```

Parameters

[out] pDesc

Type: [DXGI_OUTPUT_DESC1*](#)

A pointer to the output description (see [DXGI_OUTPUT_DESC1](#)).

Return value

Type: [HRESULT](#)

Returns a code that indicates success or failure. [S_OK](#) if successful, [DXGI_ERROR_INVALID_CALL](#) if *pDesc* is passed in as [NULL](#).

Remarks

Some scenarios do not have well-defined values for all fields in this struct. For example, if this IDXGIOutput represents a clone/duplicate set, or if the EDID has missing or invalid data. In these cases, the OS will provide some default values that correspond to a standard SDR display.

An output's reported color and luminance characteristics can adjust dynamically while the system is running due to user action or changing ambient conditions. Therefore,

apps should periodically query `IDXGIFactory::IsCurrent` and re-create their `IDXGIFactory` if it returns `FALSE`. Then re-query `GetDesc1` from the new factory's equivalent output to retrieve the newest color information.

For more details on how to write apps that react dynamically to monitor capabilities, see [Using DirectX with high dynamic range displays and Advanced Color](#).

On a high DPI desktop, `GetDesc1` returns the visualized screen size unless the app is marked high DPI aware. For info about writing DPI-aware Win32 apps, see [High DPI](#).

Requirements

Target Platform	Windows
Header	<code>dxgi1_6.h</code>
Library	<code>DXGI.lib</code>

See also

[DXGI Interfaces](#)

[IDXGIOutput6](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgicommon.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgicommon.h contains the following programming interfaces:

Structures

[DXGI_RATIONAL](#)

Represents a rational number.

[DXGI_SAMPLE_DESC](#)

Describes multi-sampling parameters for a resource.

Enumerations

[DXGI_COLOR_SPACE_TYPE](#)

Specifies color space types.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_COLOR_SPACE_TYPE enumeration (dxgicommon.h)

Article03/11/2023

Specifies color space types.

Syntax

C++

```
typedef enum DXGI_COLOR_SPACE_TYPE {
    DXGI_COLOR_SPACE_RGB_FULL_G22_NONE_P709 = 0,
    DXGI_COLOR_SPACE_RGB_FULL_G10_NONE_P709 = 1,
    DXGI_COLOR_SPACE_RGB_STUDIO_G22_NONE_P709 = 2,
    DXGI_COLOR_SPACE_RGB_STUDIO_G22_NONE_P2020 = 3,
    DXGI_COLOR_SPACE_RESERVED = 4,
    DXGI_COLOR_SPACE_YCBCR_FULL_G22_NONE_P709_X601 = 5,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P601 = 6,
    DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P601 = 7,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P709 = 8,
    DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P709 = 9,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P2020 = 10,
    DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P2020 = 11,
    DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020 = 12,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G2084_LEFT_P2020 = 13,
    DXGI_COLOR_SPACE_RGB_STUDIO_G2084_NONE_P2020 = 14,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_TOPLEFT_P2020 = 15,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G2084_TOPLEFT_P2020 = 16,
    DXGI_COLOR_SPACE_RGB_FULL_G22_NONE_P2020 = 17,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_GHLG_TOPLEFT_P2020 = 18,
    DXGI_COLOR_SPACE_YCBCR_FULL_GHLG_TOPLEFT_P2020 = 19,
    DXGI_COLOR_SPACE_RGB_STUDIO_G24_NONE_P709 = 20,
    DXGI_COLOR_SPACE_RGB_STUDIO_G24_NONE_P2020 = 21,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_LEFT_P709 = 22,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_LEFT_P2020 = 23,
    DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_TOPLEFT_P2020 = 24,
    DXGI_COLOR_SPACE_CUSTOM = 0xFFFFFFFF
};
```

Constants

DXGI_COLOR_SPACE_RGB_FULL_G22_NONE_P709

Value: 0

Property	Value
Colorspace	RGB
Range	0-255
Gamma	2.2
Siting	Image
Primaries	BT.709

This is the standard definition for sRGB.

Note

This is intended to be implemented with sRGB gamma (linear segment + 2.4 power), which is approximately aligned with a gamma 2.2 curve.

This is usually used with 8 or 10 bit color channels.

`DXGI_COLOR_SPACE_RGB_FULL_G10_NONE_P709`

Value: 1

Property	Value
Colorspace	RGB
Range	0-255
Gamma	1.0
Siting	Image
Primaries	BT.709

This is the standard definition for scRGB, and is usually used with 16 bit integer, 16 bit floating point, or 32 bit floating point color channels.

`DXGI_COLOR_SPACE_RGB_STUDIO_G22_NONE_P709`

Value: 2

Property	Value
Colorspace	RGB
Range	16-235
Gamma	2.2
Siting	Image
Primaries	BT.709

This is the standard definition for ITU-R Recommendation BT.709. Note that due to the inclusion of a linear segment, the transfer curve looks similar to a pure exponential gamma of 1.9.

This is usually used with 8 or 10 bit color channels.

`DXGI_COLOR_SPACE_RGB_STUDIO_G22_NONE_P2020`

Value: 3

Property	Value
Colorspace	RGB
Range	16-235
Gamma	2.2
Siting	Image
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_RESERVED`

Value: 4

Reserved.

`DXGI_COLOR_SPACE_YCBCR_FULL_G22_NONE_P709_X601`

Value: 5

Property	Value
Colorspace	YCbCr
Range	0-255
Gamma	2.2
Siting	Image
Primaries	BT.709
Transfer Matrix	BT.601

This definition is commonly used for JPG, and is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P601`

Value: 6

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2.2
Siting	Video
Primaries	BT.601

This definition is commonly used for MPEG2, and is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P601`

Value: 7

Property	Value
Colorspace	YCbCr
Range	0-255
Gamma	2.2
Siting	Video
Primaries	BT.601

This is sometimes used for H.264 camera capture, and is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P709`

Value: 8

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2.2
Siting	Video
Primaries	BT.709

This definition is commonly used for H.264 and HEVC, and is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P709`

Value: 9

Property	Value
Colorspace	YCbCr
Range	0-255
Gamma	2.2
Siting	Video
Primaries	BT.709

This is sometimes used for H.264 camera capture, and is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_LEFT_P2020`

Value: 10

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2.2
Siting	Video
Primaries	BT.2020

This definition may be used by HEVC, and is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_FULL_G22_LEFT_P2020`

Value: 71

Property	Value
Colorspace	YCbCr
Range	0-255
Gamma	2.2
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_RGB_FULL_G2084_NONE_P2020`

Value: 12

Property	Value
Colorspace	RGB
Range	0-255
Gamma	2084
Siting	Image
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G2084_LEFT_P2020`

Value: 73

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2084
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_RGB_STUDIO_G2084_NONE_P2020`

Value: 14

Property	Value
Colorspace	RGB
Range	16-235
Gamma	2084
Siting	Image
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G22_TOPLEFT_P2020`

Value: 75

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2.2
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G2084_TOPLEFT_P2020`

Value: 16

Property	Value
Colorspace	YCbCr
Range	16-235
Gamma	2084
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_RGB_FULL_G22_NONE_P2020`

Value: 17

Property	Value
Colorspace	RGB
Range	0-255
Gamma	2.2
Siting	Image
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_GHLG_TOPLEFT_P2020`

Value: 18

Property	Value
Colorspace	YCBCR
Range	16-235
Gamma	HLG
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_FULL_GHLG_TOPLEFT_P2020`

Value: 79

Property	Value
Colorspace	YCBCR
Range	0-255
Gamma	HLG
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_RGB_STUDIO_G24_NONE_P709`

Value: 20

Property	Value
Colorspace	RGB
Range	16-235
Gamma	2.4
Siting	Image
Primaries	BT.709

This is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_RGB_STUDIO_G24_NONE_P2020`

Value: 21

Property	Value
Colorspace	RGB
Range	16-235
Gamma	2.4
Siting	Image
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_LEFT_P709`

Value: 22

Property	Value
Colorspace	YCBCR
Range	16-235
Gamma	2.4
Siting	Video
Primaries	BT.709

This is usually used with 8, 10, or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_LEFT_P2020`

Value: 23

Property	Value
Colorspace	YCBCR
Range	16-235
Gamma	2.4
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_YCBCR_STUDIO_G24_TOPLEFT_P2020`

Value: 24

Property	Value
Colorspace	YCBCR
Range	16-235
Gamma	2.4
Siting	Video
Primaries	BT.2020

This is usually used with 10 or 12 bit color channels.

`DXGI_COLOR_SPACE_CUSTOM`

Value: `0xFFFFFFFF`

A custom color definition is used.

Remarks

This enum is used within DXGI in the [CheckColorSpaceSupport](#), [SetColorSpace1](#) and [CheckOverlayColorSpaceSupport](#) methods. It is also referenced in D3D11 video methods such as [ID3D11VideoContext1::VideoProcessorSetOutputColorSpace1](#), and D2D methods such as [ID2D1DeviceContext2::CreateImageSourceFromDxgi](#).

The following color parameters are defined:

Colorspace

Defines the color space of the color channel data.

Defined Values	Notation in color space enumeration	Comments
RGB	_RGB_	The red/green/blue color space color channel.
YCbCr	_YCbCr_	Three channel color model which splits luma (brightness) from chroma (color). YUV technically refers to analog signals and YCbCr to digital, but they are used interchangeably.

Range

Indicates which integer range corresponds to the floating point [0..1] range of the data. For video, integer YCbCr data with ranges of [16..235] or [8..247] are usually mapped to normalized YCbCr with ranges of [0..1] or [-0.5..0.5].

Defined_Values	Notation in color space numeration	Comments
8 bit: 0-255 10 bit: 0-1023 12 bit: 0-4095	_FULL_	PC desktop content and images.
8 bit:16-235 10 bit: 64-940 12 bit: 256 - 3760	_STUDIO_	Often used in video. Enables the calibration of white and black between displays.

Gamma

Defined Values	Notation in color space enumeration	Comments
1.0	_G10_	Linear light levels.
2.2	_G22_	Commonly used for sRGB and BT.709 (linear segment + 2.4).
2084	_G2084_	See SMPTE ST.2084 (Perceptual Quantization)

Siting

"Siting" indicates a horizontal or vertical shift of the chrominance channels relative to the luminance channel. "Cositing" indicates values are sited between pixels in the vertical or horizontal direction (also known as being "sited interstitially").

Defined Values	Notation in color space enumeration	Comments	For Example
Image	_NONE_	The U and V planes are aligned vertically.	MPEG1, JPG
Video	_LEFT_	Chroma samples are aligned horizontally with the luma samples, or with multiples of the luma samples. The U and V planes are aligned vertically.	MPEG2, MPEG4
Video	_TOPLEFT_	"Top left" means that the sampling point is the top left pixel (usually of a 2x2 pixel block). Chroma samples are aligned horizontally with the luma samples, or with multiples of the luma samples. Chroma samples are also aligned vertically with the luma samples, or with multiples of the luma samples.	UHD Blu-Ray

For more information on siting, refer to the [MFVideoChromaSubsampling](#) enum.

Primaries

Defined Values	Notation in color space enumeration	Comments
BT.601	_P601	Standard defining digital encoding of SDTV video.

BT.709	_P709	Standard defining digital encoding of HDTV video.
BT.2020	_P2020	Standard defining ultra-high definition television (UHDTV).

Transfer Matrix

In most cases, the transfer matrix can be determined from the primaries. For some cases it must be explicitly specified as described below:

Defined Values	Notation in color space enumeration	Comments
BT.601	_X601	Standard defining digital encoding of SDTV video.
BT.709	_X709	Standard defining digital encoding of HDTV video.
BT.2020	_X2020	Standard defining ultra-high definition television (UHDTV).

Subsampling and the layout of the color channels are inferred from the surface format.

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Header	dxgicommon.h (include DXGIType.h)

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_RATIONAL structure (dxgicommon.h)

Article 04/02/2021

Represents a rational number.

Syntax

C++

```
typedef struct DXGI_RATIONAL {  
    UINT Numerator;  
    UINT Denominator;  
} DXGI_RATIONAL;
```

Members

Numerator

Type: [UINT](#)

An unsigned integer value representing the top of the rational number.

Denominator

Type: [UINT](#)

An unsigned integer value representing the bottom of the rational number.

Remarks

This structure is a member of the [DXGI_MODE_DESC](#) structure.

The **DXGI_RATIONAL** structure operates under the following rules:

- 0/0 is legal and will be interpreted as 0/1.
- 0/anything is interpreted as zero.
- If you are representing a whole number, the denominator should be 1.

Requirements

Header

dxgicommon.h (include DXGI.h)

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_SAMPLE_DESC structure (dxgicommon.h)

Article04/02/2021

Describes multi-sampling parameters for a resource.

Syntax

C++

```
typedef struct DXGI_SAMPLE_DESC {  
    UINT Count;  
    UINT Quality;  
} DXGI_SAMPLE_DESC;
```

Members

Count

Type: [UINT](#)

The number of multisamples per pixel.

Quality

Type: [UINT](#)

The image quality level. The higher the quality, the lower the performance. The valid range is between zero and one less than the level returned by [ID3D10Device::CheckMultisampleQualityLevels](#) for Direct3D 10 or [ID3D11Device::CheckMultisampleQualityLevels](#) for Direct3D 11.

For Direct3D 10.1 and Direct3D 11, you can use two special quality level values. For more information about these quality level values, see Remarks.

Remarks

This structure is a member of the [DXGI_SWAP_CHAIN_DESC1](#) structure.

The default sampler mode, with no anti-aliasing, has a count of 1 and a quality level of 0.

If multi-sample antialiasing is being used, all bound render targets and depth buffers must have the same sample counts and quality levels.

Differences between Direct3D 10.0 and Direct3D 10.1 and between Direct3D 10.0 and Direct3D 11:
Direct3D 10.1 has defined two standard quality levels:

D3D10_STANDARD_MULTISAMPLE_PATTERN and **D3D10_CENTER_MULTISAMPLE_PATTERN** in
the **D3D10_STANDARD_MULTISAMPLE_QUALITY_LEVELS** enumeration in D3D10_1.h.

Direct3D 11 has defined two standard quality levels:

D3D11_STANDARD_MULTISAMPLE_PATTERN and **D3D11_CENTER_MULTISAMPLE_PATTERN** in
the **D3D11_STANDARD_MULTISAMPLE_QUALITY_LEVELS** enumeration in D3D11.h.

Requirements

Header	dxicommon.h (include DXGI.h)
--------	------------------------------

See also

[DXGI Structures](#)

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

dxgidebug.h header

Article 01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgidebug.h contains the following programming interfaces:

Interfaces

[IDXGIDebug](#)

This interface controls debug settings, and can only be used if the debug layer is turned on.

[IDXGIDebug1](#)

Controls debug settings for Microsoft DirectX Graphics Infrastructure (DXGI). You can use the IDXGIDebug1 interface in Windows Store apps.

[IDXGIIInfoQueue](#)

This interface controls the debug information queue, and can only be used if the debug layer is turned on.

Functions

[DXGIGetDebugInterface](#)

Retrieves a debugging interface.

Structures

[DXGI_INFO_QUEUE_FILTER](#)

Describes a debug message filter, which contains lists of message types to allow and deny.

[DXGI_INFO_QUEUE_FILTER_DESC](#)

Describes the types of messages to allow or deny to pass through a filter.

[DXGI_INFO_QUEUE_MESSAGE](#)

Describes a debug message in the information queue.

Enumerations

[DXGI_DEBUG_RLO_FLAGS](#)

Flags used with ReportLiveObjects to specify the amount of info to report about an object's lifetime.

[DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)

Values that specify categories of debug messages.

[DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)

Values that specify debug message severity levels for an information queue.

Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

DXGI_DEBUG_RLO_FLAGS enumeration (dxgidebug.h)

Article 01/31/2022

Flags used with [ReportLiveObjects](#) to specify the amount of info to report about an object's lifetime.

Syntax

C++

```
typedef enum DXGI_DEBUG_RLO_FLAGS {
    DXGI_DEBUG_RLO_SUMMARY = 0x1,
    DXGI_DEBUG_RLO_DETAIL = 0x2,
    DXGI_DEBUG_RLO_IGNORE_INTERNAL = 0x4,
    DXGI_DEBUG_RLO_ALL = 0x7
};
```

Constants

`DXGI_DEBUG_RLO_SUMMARY`

Value: *0x1*

A flag that specifies to obtain a summary about an object's lifetime.

`DXGI_DEBUG_RLO_DETAIL`

Value: *0x2*

A flag that specifies to obtain detailed info about an object's lifetime.

`DXGI_DEBUG_RLO_IGNORE_INTERNAL`

Value: *0x4*

This flag indicates to ignore objects which have no external refcounts keeping them alive.

D3D objects are printed using an external refcount and an internal refcount.

Typically, all objects are printed.

This flag means ignore the objects whose external refcount is 0, because the application is not responsible for keeping them alive.

`DXGI_DEBUG_RLO_ALL`

Value: *0x7*

A flag that specifies to obtain both a summary and detailed info about an object's lifetime.

Remarks

Use this enumeration with [IDXGIDebug::ReportLiveObjects](#).

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Header	dkgidebug.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_INFO_QUEUE_FILTER structure (dxgidebug.h)

Article 10/05/2021

Describes a debug message filter, which contains lists of message types to allow and deny.

Syntax

C++

```
typedef struct DXGI_INFO_QUEUE_FILTER {
    DXGI_INFO_QUEUE_FILTER_DESC AllowList;
    DXGI_INFO_QUEUE_FILTER_DESC DenyList;
} DXGI_INFO_QUEUE_FILTER;
```

Members

`AllowList`

A [DXGI_INFO_QUEUE_FILTER_DESC](#) structure that describes the types of messages to allow.

`DenyList`

A [DXGI_INFO_QUEUE_FILTER_DESC](#) structure that describes the types of messages to deny.

Remarks

Use with an [IDXGIFInfoQueue](#) interface.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Header	dkgidebug.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

DXGI_INFO_QUEUE_FILTER_DESC structure (dxgidebug.h)

Article 10/05/2021

Describes the types of messages to allow or deny to pass through a filter.

Syntax

C++

```
typedef struct DXGI_INFO_QUEUE_FILTER_DESC {
    UINT NumCategories;
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY *pCategoryList;
    UINT NumSeverities;
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY *pSeverityList;
    UINT NumIDs;
    DXGI_INFO_QUEUE_MESSAGE_ID     *pIDList;
} DXGI_INFO_QUEUE_FILTER_DESC;
```

Members

NumCategories

The number of message categories to allow or deny.

pCategoryList

An array of [DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#) enumeration values that describe the message categories to allow or deny. The array must have at least **NumCategories** number of elements.

NumSeverities

The number of message severity levels to allow or deny.

pSeverityList

An array of [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#) enumeration values that describe the message severity levels to allow or deny. The array must have at least **NumSeverities** number of elements.

NumIDs

The number of message IDs to allow or deny.

pIDList

An array of integers that represent the message IDs to allow or deny. The array must have at least **NumIDs** number of elements.

Remarks

This structure is a member of the [DXGI_INFO_QUEUE_FILTER](#) structure.

This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Header	dxi gidebug.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_INFO_QUEUE_MESSAGE structure (dxgidebug.h)

Article 10/05/2021

Describes a debug message in the information queue.

Syntax

C++

```
typedef struct DXGI_INFO_QUEUE_MESSAGE {
    DXGI_DEBUG_ID                     Producer;
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY Category;
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY Severity;
    DXGI_INFO_QUEUE_MESSAGE_ID        ID;
    const char*                        pDescription;
    SIZE_T                             DescriptionByteLength;
} DXGI_INFO_QUEUE_MESSAGE;
```

Members

Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that produced the message.

Category

A [DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)-typed value that specifies the category of the message.

Severity

A [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)-typed value that specifies the severity of the message.

ID

An integer that uniquely identifies the message.

pDescription

The message string.

DescriptionByteLength

The length of the message string at `pDescription`, in bytes.

Remarks

`IDXGIIInfoQueue::GetMessage` returns a pointer to this structure.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Header	<code>dxiidebug.h</code>

See also

[DXGI Structures](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

DXGI_INFO_QUEUE_MESSAGE_CATEGORY enumeration (dxgidebug.h)

Article 01/31/2022

Values that specify categories of debug messages.

Syntax

C++

```
typedef enum DXGI_INFO_QUEUE_MESSAGE_CATEGORY {
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_UNKNOWN = 0,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_MISCELLANEOUS,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_INITIALIZATION,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_CLEANUP,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_COMPILATION,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_CREATION,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_SETTING,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_GETTING,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_RESOURCE_MANIPULATION,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_EXECUTION,
    DXGI_INFO_QUEUE_MESSAGE_CATEGORY_SHADER
};
```

Constants

DXGI_INFO_QUEUE_MESSAGE_CATEGORY_UNKNOWN

Value: 0

Unknown category.

DXGI_INFO_QUEUE_MESSAGE_CATEGORY_MISCELLANEOUS

Miscellaneous category.

DXGI_INFO_QUEUE_MESSAGE_CATEGORY_INITIALIZATION

Initialization category.

DXGI_INFO_QUEUE_MESSAGE_CATEGORY_CLEANUP

Cleanup category.

DXGI_INFO_QUEUE_MESSAGE_CATEGORY_COMPILATION

Compilation category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_CREATION`

State creation category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_SETTING`

State setting category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_STATE_GETTING`

State getting category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_RESOURCE_MANIPULATION`

Resource manipulation category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_EXECUTION`

Execution category.

`DXGI_INFO_QUEUE_MESSAGE_CATEGORY_SHADER`

Shader category.

Remarks

Use this enumeration when you call [IDXGIFilter::GetMessage](#) to retrieve a message and when you call [IDXGIFilter::AddMessage](#) to add a message. When you create an info queue filter, you can use these values to allow or deny any categories of messages to pass through the storage and retrieval filters.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client Windows 8 [desktop apps | UWP apps]

Minimum supported server Windows Server 2012 [desktop apps | UWP apps]

Header dxgidebug.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_INFO_QUEUE_MESSAGE_SEVERITY enumeration (dxgidebug.h)

Article 01/31/2022

Values that specify debug message severity levels for an information queue.

Syntax

C++

```
typedef enum DXGI_INFO_QUEUE_MESSAGE_SEVERITY {
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY_CORRUPTION = 0,
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY_ERROR,
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY_WARNING,
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY_INFO,
    DXGI_INFO_QUEUE_MESSAGE_SEVERITY_MESSAGE
} ;
```

Constants

`DXGI_INFO_QUEUE_MESSAGE_SEVERITY_CORRUPTION`

Value: 0

Defines some type of corruption that has occurred.

`DXGI_INFO_QUEUE_MESSAGE_SEVERITY_ERROR`

Defines an error message.

`DXGI_INFO_QUEUE_MESSAGE_SEVERITY_WARNING`

Defines a warning message.

`DXGI_INFO_QUEUE_MESSAGE_SEVERITY_INFO`

Defines an information message.

`DXGI_INFO_QUEUE_MESSAGE_SEVERITY_MESSAGE`

Defines a message other than corruption, error, warning, or information.

Remarks

Use this enumeration when you call [IDXGIFInfoQueue::GetMessage](#) to retrieve a message and when you call [IDXGIFInfoQueue::AddMessage](#) to add a message. Also, use this

enumeration with [IDXGIFInfoQueue::AddApplicationMessage](#).

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Header	dwgidebug.h

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGIGetDebugInterface function (dxgidebug.h)

Article 10/05/2021

Retrieves a debugging interface.

Syntax

C++

```
HRESULT DXGIGetDebugInterface(  
    REFIID riid,  
    void    **ppDebug  
>;
```

Parameters

riid

The globally unique identifier (GUID) of the requested interface type.

ppDebug

A pointer to a buffer that receives a pointer to the debugging interface.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

[IDXGIDebug](#) and [IDXGILInfoQueue](#) are debugging interfaces.

To access [DXGIGetDebugInterface](#), call the [GetProcAddress](#) function to get Dxgidebug.dll and the [GetProcAddress](#) function to get the address of [DXGIGetDebugInterface](#).
Windows 8.1: Starting in Windows 8.1, Windows Store apps call the [DXGIGetDebugInterface1](#) function to get an [IDXGIDebug1](#) interface.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dwgidebug.h
DLL	Dwgidebug.dll

See also

[DXGI Functions](#)

[IDXGIDebug](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDebug interface (dxgidebug.h)

Article 10/05/2021

This interface controls debug settings, and can only be used if the debug layer is turned on.

Inheritance

The **IDXGIDebug** interface inherits from the [IUnknown](#) interface. **IDXGIDebug** also has these types of members:

Methods

The **IDXGIDebug** interface has these methods.

[IDXGIDebug::ReportLiveObjects](#)

Reports info about the lifetime of an object or objects.

Remarks

This interface is obtained by calling the [DXGIGetDebugInterface](#) function.

For more info about the debug layer, see [Debug Layer](#).

Windows Phone 8: This API is supported.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dkgidebug.h

See also

[DXGI Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIDebug::ReportLiveObjects method (dxgidebug.h)

Article 10/05/2021

Reports info about the lifetime of an object or objects.

See the [Memory management sample](#).

Syntax

C++

```
HRESULT ReportLiveObjects(
    GUID                 apiid,
    DXGI_DEBUG_RLO_FLAGS flags
);
```

Parameters

`apiid`

The globally unique identifier (GUID) of the object or objects to get info about. Use one of the [DXGI_DEBUG_ID](#) GUIDs.

`flags`

A [DXGI_DEBUG_RLO_FLAGS](#)-typed value that specifies the amount of info to report.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIDebug](#)

[Memory management sample ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDebug1 interface (dxgidebug.h)

Article 10/05/2021

Controls debug settings for Microsoft DirectX Graphics Infrastructure (DXGI). You can use the [IDXGIDebug1](#) interface in Windows Store apps.

Inheritance

The [IDXGIDebug1](#) interface inherits from [IDXGIDebug](#). [IDXGIDebug1](#) also has these types of members:

Methods

The [IDXGIDebug1](#) interface has these methods.

IDXGIDebug1::DisableLeakTrackingForThread
Stops tracking leaks for the current thread.
IDXGIDebug1::EnableLeakTrackingForThread
Starts tracking leaks for the current thread.
IDXGIDebug1::IsLeakTrackingEnabledForThread
Gets a value indicating whether leak tracking is turned on for the current thread.

Remarks

Call the [DXGIGetDebugInterface1](#) function to obtain the [IDXGIDebug1](#) interface.

The [IDXGIDebug1](#) interface can be used only if the debug layer is turned on. For more info, see [Debug Layer](#).

Requirements

Minimum supported client	Windows 8.1 [desktop apps UWP apps]
--------------------------	---------------------------------------

Minimum supported server	Windows Server 2012 R2 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h

See also

[DXGI Interfaces](#)

[DXGIGetDebugInterface1](#)

[IDXGIDebug](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIDebug1::DisableLeakTrackingForThread method (dxgidebug.h)

Article 10/05/2021

Stops tracking leaks for the current thread.

Syntax

C++

```
void DisableLeakTrackingForThread();
```

Return value

None

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[DXGIGetDebugInterface1](#)

[IDXGIDebug1](#)

Feedback



Was this page helpful?  

Get help at Microsoft Q&A

IDXGIDebug1::EnableLeakTrackingForThread method (dxgidebug.h)

Article 10/05/2021

Starts tracking leaks for the current thread.

Syntax

C++

```
void EnableLeakTrackingForThread();
```

Return value

None

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[DXGIGetDebugInterface1](#)

[IDXGIDebug1](#)

Feedback



Was this page helpful?  

Get help at Microsoft Q&A

IDXGIDebug1::IsLeakTrackingEnabledForThread method (dxgidebug.h)

Article 10/05/2021

Gets a value indicating whether leak tracking is turned on for the current thread.

Syntax

C++

```
BOOL IsLeakTrackingEnabledForThread();
```

Return value

TRUE if leak tracking is turned on for the current thread; otherwise, FALSE.

Requirements

Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[DXGIGetDebugInterface1](#)

[IDXGIDebug1](#)

Feedback



Was this page helpful?  

Get help at Microsoft Q&A

IDXGIIInfoQueue interface (dxgidebug.h)

Article 10/05/2021

This interface controls the debug information queue, and can only be used if the debug layer is turned on.

Inheritance

The **IDXGIIInfoQueue** interface inherits from the [IUnknown](#) interface. **IDXGIIInfoQueue** also has these types of members:

Methods

The **IDXGIIInfoQueue** interface has these methods.

IDXGIIInfoQueue::AddApplicationMessage
Adds a user-defined message to the message queue and sends that message to the debug output.
IDXGIIInfoQueue::AddMessage
Adds a debug message to the message queue and sends that message to the debug output.
IDXGIIInfoQueue::AddRetrievalFilterEntries
Adds retrieval filters to the top of the retrieval-filter stack.
IDXGIIInfoQueue::AddStorageFilterEntries
Adds storage filters to the top of the storage-filter stack.
IDXGIIInfoQueue::ClearRetrievalFilter
Removes a retrieval filter from the top of the retrieval-filter stack.
IDXGIIInfoQueue::ClearStorageFilter
Removes a storage filter from the top of the storage-filter stack.
IDXGIIInfoQueue::ClearStoredMessages
Clears all messages from the message queue.

[IDXGInfoQueue::GetBreakOnCategory](#)

Determines whether the break on a message category is turned on or off.

[IDXGInfoQueue::GetBreakOnID](#)

Determines whether the break on a message identifier is turned on or off.

[IDXGInfoQueue::GetBreakOnSeverity](#)

Determines whether the break on a message severity level is turned on or off.

[IDXGInfoQueue::GetMessage](#)

Gets a message from the message queue.

[IDXGInfoQueue::GetMessageCountLimit](#)

Gets the maximum number of messages that can be added to the message queue.

[IDXGInfoQueue::GetMuteDebugOutput](#)

Determines whether the debug output is turned on or off.

[IDXGInfoQueue::GetNumMessagesAllowedByStorageFilter](#)

Gets the number of messages that a storage filter allowed to pass through.

[IDXGInfoQueue::GetNumMessagesDeniedByStorageFilter](#)

Gets the number of messages that were denied passage through a storage filter.

[IDXGInfoQueue::GetNumMessagesDiscardedByMessageCountLimit](#)

Gets the number of messages that were discarded due to the message count limit.

[IDXGInfoQueue::GetNumStoredMessages](#)

Gets the number of messages currently stored in the message queue.

[IDXGInfoQueue::GetNumStoredMessagesAllowedByRetrievalFilters](#)

Gets the number of messages that can pass through a retrieval filter.

[IDXGInfoQueue::GetRetrievalFilter](#)

Gets the retrieval filter at the top of the retrieval-filter stack.

[IDXGInfoQueue::GetRetrievalFilterStackSize](#)

Gets the size of the retrieval-filter stack in bytes.

[IDXGInfoQueue::GetStorageFilter](#)

Gets the storage filter at the top of the storage-filter stack.

[IDXGInfoQueue::GetStorageFilterStackSize](#)

Gets the size of the storage-filter stack in bytes.

[IDXGInfoQueue::PopRetrievalFilter](#)

Pops a retrieval filter from the top of the retrieval-filter stack.

[IDXGInfoQueue::PopStorageFilter](#)

Pops a storage filter from the top of the storage-filter stack.

[IDXGInfoQueue::PushCopyOfRetrievalFilter](#)

Pushes a copy of the retrieval filter that is currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

[IDXGInfoQueue::PushCopyOfStorageFilter](#)

Pushes a copy of the storage filter that is currently on the top of the storage-filter stack onto the storage-filter stack.

[IDXGInfoQueue::PushDenyAllRetrievalFilter](#)

Pushes a deny-all retrieval filter onto the retrieval-filter stack.

[IDXGInfoQueue::PushDenyAllStorageFilter](#)

Pushes a deny-all storage filter onto the storage-filter stack.

[IDXGInfoQueue::PushEmptyRetrievalFilter](#)

Pushes an empty retrieval filter onto the retrieval-filter stack.

[IDXGInfoQueue::PushEmptyStorageFilter](#)

Pushes an empty storage filter onto the storage-filter stack.

[IDXGInfoQueue::PushRetrievalFilter](#)

Pushes a retrieval filter onto the retrieval-filter stack.

[IDXGInfoQueue::PushStorageFilter](#)

Pushes a storage filter onto the storage-filter stack.

[IDXGInfoQueue::SetBreakOnCategory](#)

Sets a message category to break on when a message with that category passes through the storage filter.

[IDXGInfoQueue::SetBreakOnID](#)

Sets a message identifier to break on when a message with that identifier passes through the storage filter.

[IDXGInfoQueue::SetBreakOnSeverity](#)

Sets a message severity level to break on when a message with that severity level passes through the storage filter.

[IDXGInfoQueue::SetMessageCountLimit](#)

Sets the maximum number of messages that can be added to the message queue.

[IDXGInfoQueue::SetMuteDebugOutput](#)

Turns the debug output on or off.

Remarks

This interface is obtained by calling the [DXGIGetDebugInterface](#) function.

For more info about the debug layer, see [Debug Layer](#).

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h

See also

[DXGI Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::AddApplicationMessage method (dxgidebug.h)

Article 10/13/2021

Adds a user-defined message to the message queue and sends that message to the debug output.

Syntax

C++

```
HRESULT AddApplicationMessage(
    [in] DXGI_INFO_QUEUE_MESSAGE_SEVERITY Severity,
    [in] LPCSTR pDescription
);
```

Parameters

[in] Severity

A [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)-typed value that specifies the severity of the message.

[in] pDescription

The message string.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::AddMessage method (dxgidebug.h)

Article 10/13/2021

Adds a debug message to the message queue and sends that message to the debug output.

Syntax

C++

```
HRESULT AddMessage(
    [in] DXGI_DEBUG_ID                 Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_CATEGORY Category,
    [in] DXGI_INFO_QUEUE_MESSAGE_SEVERITY Severity,
    [in] DXGI_INFO_QUEUE_MESSAGE_ID      ID,
    [in] LPCSTR                         pDescription
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that produced the message.

[in] Category

A [DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)-typed value that specifies the category of the message.

[in] Severity

A [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)-typed value that specifies the severity of the message.

[in] ID

An integer that uniquely identifies the message.

[in] pDescription

The message string.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGILInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::AddRetrievalFilterEntries method (dxgidebug.h)

Article 10/13/2021

Adds retrieval filters to the top of the retrieval-filter stack.

Syntax

C++

```
HRESULT AddRetrievalFilterEntries(
    [in] DXGI_DEBUG_ID           Producer,
    [in] DXGI_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that produced the filters.

[in] pFilter

An array of [DXGI_INFO_QUEUE_FILTER](#) structures that describe the filters.

Return value

Returns [S_OK](#) if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::AddStorageFilterEntries method (dxgidebug.h)

Article 10/13/2021

Adds storage filters to the top of the storage-filter stack.

Syntax

C++

```
HRESULT AddStorageFilterEntries(
    [in] DXGI_DEBUG_ID           Producer,
    [in] DXGI_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that produced the filters.

[in] pFilter

An array of [DXGI_INFO_QUEUE_FILTER](#) structures that describe the filters.

Return value

Returns [S_OK](#) if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::ClearRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Removes a retrieval filter from the top of the retrieval-filter stack.

Syntax

C++

```
void ClearRetrievalFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that removes the filter.

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::ClearStorageFilter method (dxgidebug.h)

Article 10/13/2021

Removes a storage filter from the top of the storage-filter stack.

Syntax

C++

```
void ClearStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that removes the filter.

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::ClearStoredMessages method (dxgidebug.h)

Article 10/13/2021

Clears all messages from the message queue.

Syntax

C++

```
void ClearStoredMessages(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that clears the messages.

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetBreakOnCategory method (dxgidebug.h)

Article 10/13/2021

Determines whether the break on a message category is turned on or off.

Syntax

C++

```
BOOL GetBreakOnCategory(
    [in] DXGI_DEBUG_ID                 Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_CATEGORY Category
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the breaking status.

[in] Category

A [DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)-typed value that specifies the category of the message.

Return value

Returns a Boolean value that specifies whether this category of breaking condition is turned on or off (**TRUE** for on, **FALSE** for off).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetBreakOnID method (dxgidebug.h)

Article 10/13/2021

Determines whether the break on a message identifier is turned on or off.

Syntax

C++

```
BOOL GetBreakOnID(  
    [in] DXGI_DEBUG_ID             Producer,  
    [in] DXGI_INFO_QUEUE_MESSAGE_ID ID  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the breaking status.

[in] ID

An integer value that specifies the identifier of the message.

Return value

Returns a Boolean value that specifies whether this break on a message identifier is turned on or off (**TRUE** for on, **FALSE** for off).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetBreakOnSeverity method (dxgidebug.h)

Article 10/13/2021

Determines whether the break on a message severity level is turned on or off.

Syntax

C++

```
BOOL GetBreakOnSeverity(
    [in] DXGI_DEBUG_ID                 Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_SEVERITY Severity
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the breaking status.

[in] Severity

A [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)-typed value that specifies the severity of the message.

Return value

Returns a Boolean value that specifies whether this severity of breaking condition is turned on or off (**TRUE** for on, **FALSE** for off).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetMessage method (dxgidebug.h)

Article 10/13/2021

Gets a message from the message queue.

Syntax

C++

```
HRESULT GetMessage(
    [in]           DXGI_DEBUG_ID           Producer,
    [in]           UINT64                 MessageIndex,
    [out, optional] DXGI_INFO_QUEUE_MESSAGE *pMessage,
    [in, out]        SIZE_T                 *pMessageByteLength
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the message.

[in] MessageIndex

An index into the message queue after an optional retrieval filter has been applied. This can be between 0 and the number of messages in the message queue that pass through the retrieval filter. Call

[IDXGIIInfoQueue::GetNumStoredMessagesAllowedByRetrievalFilters](#) to obtain this number. 0 is the message at the beginning of the message queue.

[out, optional] pMessage

A pointer to a [DXGI_INFO_QUEUE_MESSAGE](#) structure that describes the message.

[in, out] pMessageByteLength

A pointer to a variable that receives the size, in bytes, of the message description that *pMessage* points to. This size includes the size of the [DXGI_INFO_QUEUE_MESSAGE](#) structure in bytes.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

This method doesn't remove any messages from the message queue.

This method gets a message from the message queue after an optional retrieval filter has been applied.

Call this method twice to retrieve a message, first to obtain the size of the message and second to get the message. Here is a typical example:

```
// Get the size of the message.  
SIZE_T messageLength = 0;  
HRESULT hr = pInfoQueue->GetMessage(DXGI_DEBUG_ALL, 0, NULL,  
&messageLength);  
if(hr == S_FALSE){  
  
    // Allocate space and get the message.  
    DXGI_INFO_QUEUE_MESSAGE * pMessage =  
    (DXGI_INFO_QUEUE_MESSAGE*)malloc(messageLength);  
    hr = pInfoQueue->GetMessage(DXGI_DEBUG_ALL, 0, pMessage,  
&messageLength);  
  
    // Do something with the message and free it  
    if(hr == S_OK){  
  
        // ...  
        // ...  
        // ...  
        free(pMessage);  
    }  
}
```

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetMessageCountLimit method (dxgidebug.h)

Article 10/13/2021

Gets the maximum number of messages that can be added to the message queue.

Syntax

C++

```
UINT64 GetMessageCountLimit(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the maximum number of messages that can be added to the queue. -1 means no limit.

Remarks

When the number of messages in the message queue reaches the maximum limit, new messages coming in push old messages out.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetMuteDebugOutput method (dxgidebug.h)

Article 10/13/2021

Determines whether the debug output is turned on or off.

Syntax

C++

```
BOOL GetMuteDebugOutput(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the mute status.

Return value

Returns a Boolean value that specifies whether the debug output is turned on or off (TRUE for on, FALSE for off).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetNumMessagesAllowedByStorageFilter method (dxgidebug.h)

Article 10/13/2021

Gets the number of messages that a storage filter allowed to pass through.

Syntax

C++

```
UINT64 GetNumMessagesAllowedByStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the number of messages allowed by a storage filter.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetNumMessagesDeniedByStorageFilter method (dxgidebug.h)

Article 10/13/2021

Gets the number of messages that were denied passage through a storage filter.

Syntax

C++

```
UINT64 GetNumMessagesDeniedByStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the number of messages denied by a storage filter.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetNumMessagesDiscardedByMessageCountLimit method (dxgidebug.h)

Article 10/13/2021

Gets the number of messages that were discarded due to the message count limit.

Syntax

C++

```
UINT64 GetNumMessagesDiscardedByMessageCountLimit(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the number of messages that were discarded.

Remarks

Get and set the message count limit with [IDXGIIInfoQueue::GetMessageCountLimit](#) and [IDXGIIInfoQueue::SetMessageCountLimit](#), respectively.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetNumStoredMessages method (dxgidebug.h)

Article 10/13/2021

Gets the number of messages currently stored in the message queue.

Syntax

C++

```
UINT64 GetNumStoredMessages(  
    [in] DXGI_DEBUG_ID Producer  
>;
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the number of messages currently stored in the message queue.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetNumStoredMessagesAllowedByRetrievalFilters method (dxgidebug.h)

Article 10/13/2021

Gets the number of messages that can pass through a retrieval filter.

Syntax

C++

```
UINT64 GetNumStoredMessagesAllowedByRetrievalFilters(
    [in] DXGI_DEBUG_ID Producer
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the number.

Return value

Returns the number of messages that can pass through a retrieval filter.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client

Windows 8 [desktop apps | UWP apps]

Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Gets the retrieval filter at the top of the retrieval-filter stack.

Syntax

C++

```
HRESULT GetRetrievalFilter(
    [in]           DXGI_DEBUG_ID          Producer,
    [out, optional] DXGI_INFO_QUEUE_FILTER *pFilter,
    [in, out]       SIZE_T              *pFilterByteLength
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the filter.

[out, optional] pFilter

A pointer to a [DXGI_INFO_QUEUE_FILTER](#) structure that describes the filter.

[in, out] pFilterByteLength

A pointer to a variable that receives the size, in bytes, of the filter description to which *pFilter* points. If *pFilter* is **NULL**, **GetRetrievalFilter** outputs the size of the retrieval filter.

Return value

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetRetrievalFilterStack Size method (dxgidebug.h)

Article 10/13/2021

Gets the size of the retrieval-filter stack in bytes.

Syntax

C++

```
UINT GetRetrievalFilterStackSize(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the size.

Return value

Returns the size of the retrieval-filter stack in bytes.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetStorageFilter method (dxgidebug.h)

Article 10/13/2021

Gets the storage filter at the top of the storage-filter stack.

Syntax

C++

```
HRESULT GetStorageFilter(
    [in]           DXGI_DEBUG_ID          Producer,
    [out, optional] DXGI_INFO_QUEUE_FILTER *pFilter,
    [in, out]       SIZE_T              *pFilterByteLength
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the filter.

[out, optional] pFilter

A pointer to a [DXGI_INFO_QUEUE_FILTER](#) structure that describes the filter.

[in, out] pFilterByteLength

A pointer to a variable that receives the size, in bytes, of the filter description to which *pFilter* points. If *pFilter* is **NULL**, **GetStorageFilter** outputs the size of the storage filter.

Return value

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::GetStorageFilterStackSize method (dxgidebug.h)

Article 10/13/2021

Gets the size of the storage-filter stack in bytes.

Syntax

C++

```
UINT GetStorageFilterStackSize(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the size.

Return value

Returns the size of the storage-filter stack in bytes.

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PopRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Pops a retrieval filter from the top of the retrieval-filter stack.

Syntax

C++

```
void PopRetrievalFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pops the filter.

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PopStorageFilter method (dxgidebug.h)

Article 10/13/2021

Pops a storage filter from the top of the storage-filter stack.

Syntax

C++

```
void PopStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pops the filter.

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]

Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushCopyOfRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a copy of the retrieval filter that is currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushCopyOfRetrievalFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the copy of the retrieval filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushCopyOfStorageFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a copy of the storage filter that is currently on the top of the storage-filter stack onto the storage-filter stack.

Syntax

C++

```
HRESULT PushCopyOfStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the copy of the storage filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushDenyAllRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a deny-all retrieval filter onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushDenyAllRetrievalFilter(
    [in] DXGI_DEBUG_ID Producer
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the deny-all retrieval filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

A deny-all retrieval filter prevents all messages from passing through.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushDenyAllStorageFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a deny-all storage filter onto the storage-filter stack.

Syntax

C++

```
HRESULT PushDenyAllStorageFilter(
    [in] DXGI_DEBUG_ID Producer
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

A deny-all storage filter prevents all messages from passing through.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushEmptyRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Pushes an empty retrieval filter onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushEmptyRetrievalFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the empty retrieval filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

An empty retrieval filter allows all messages to pass through.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushEmptyStorageFilter method (dxgidebug.h)

Article 10/13/2021

Pushes an empty storage filter onto the storage-filter stack.

Syntax

C++

```
HRESULT PushEmptyStorageFilter(  
    [in] DXGI_DEBUG_ID Producer  
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the empty storage filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

An empty storage filter allows all messages to pass through.

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushRetrievalFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a retrieval filter onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushRetrievalFilter(
    [in] DXGI_DEBUG_ID           Producer,
    [in] DXGI_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the filter.

[in] pFilter

A pointer to a [DXGI_INFO_QUEUE_FILTER](#) structure that describes the filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::PushStorageFilter method (dxgidebug.h)

Article 10/13/2021

Pushes a storage filter onto the storage-filter stack.

Syntax

C++

```
HRESULT PushStorageFilter(
    [in] DXGI_DEBUG_ID           Producer,
    [in] DXGI_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that pushes the filter.

[in] pFilter

A pointer to a [DXGI_INFO_QUEUE_FILTER](#) structure that describes the filter.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::SetBreakOnCategory method (dxgidebug.h)

Article 10/13/2021

Sets a message category to break on when a message with that category passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnCategory(
    [in] DXGI_DEBUG_ID                 Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_CATEGORY Category,
    [in] BOOL                          bEnable
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that sets the breaking condition.

[in] Category

A [DXGI_INFO_QUEUE_MESSAGE_CATEGORY](#)-typed value that specifies the category of the message.

[in] bEnable

A Boolean value that specifies whether **SetBreakOnCategory** turns on or off this breaking condition (**TRUE** for on, **FALSE** for off).

Return value

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::SetBreakOnID method (dxgidebug.h)

Article 10/13/2021

Sets a message identifier to break on when a message with that identifier passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnID(
    [in] DXGI_DEBUG_ID             Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_ID ID,
    [in] BOOL                      bEnable
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that sets the breaking condition.

[in] ID

An integer value that specifies the identifier of the message.

[in] bEnable

A Boolean value that specifies whether **SetBreakOnID** turns on or off this breaking condition (**TRUE** for on, **FALSE** for off).

Return value

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::SetBreakOnSeverity method (dxgidebug.h)

Article 10/13/2021

Sets a message severity level to break on when a message with that severity level passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnSeverity(
    [in] DXGI_DEBUG_ID                 Producer,
    [in] DXGI_INFO_QUEUE_MESSAGE_SEVERITY Severity,
    [in] BOOL                          bEnable
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that sets the breaking condition.

[in] Severity

A [DXGI_INFO_QUEUE_MESSAGE_SEVERITY](#)-typed value that specifies the severity of the message.

[in] bEnable

A Boolean value that specifies whether **SetBreakOnSeverity** turns on or off this breaking condition (**TRUE** for on, **FALSE** for off).

Return value

Returns **S_OK** if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::SetMessageCountLimit method (dxgidebug.h)

Article 10/13/2021

Sets the maximum number of messages that can be added to the message queue.

Syntax

C++

```
HRESULT SetMessageCountLimit(
    [in] DXGI_DEBUG_ID Producer,
    [in] UINT64        MessageCountLimit
);
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that sets the limit on the number of messages.

[in] MessageCountLimit

The maximum number of messages that can be added to the queue. -1 means no limit.

Return value

Returns S_OK if successful; an error code otherwise. For a list of error codes, see [DXGI_ERROR](#).

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGIFInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

IDXGIIInfoQueue::SetMuteDebugOutput method (dxgidebug.h)

Article 10/13/2021

Turns the debug output on or off.

Syntax

C++

```
void SetMuteDebugOutput(  
    [in] DXGI_DEBUG_ID Producer,  
    [in] BOOL          bMute  
>;
```

Parameters

[in] Producer

A [DXGI_DEBUG_ID](#) value that identifies the entity that gets the mute status.

[in] bMute

A Boolean value that specifies whether to turn the debug output on or off (**TRUE** for on, **FALSE** for off).

Return value

None

Remarks

Note This API requires the Windows Software Development Kit (SDK) for Windows 8.

Requirements

Minimum supported client	Windows 8 [desktop apps UWP apps]
Minimum supported server	Windows Server 2012 [desktop apps UWP apps]
Target Platform	Windows
Header	dxgidebug.h
DLL	DXGIDebug.dll

See also

[IDXGInfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

dxgiformat.h header

Article01/24/2023

This header is used by DXGI. For more information, see:

- [DXGI](#)

dxgiformat.h contains the following programming interfaces:

Enumerations

[DXGI_FORMAT](#)

Resource data formats, including fully-typed and typeless formats. A list of modifiers at the bottom of the page more fully describes each format type.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

DXGI_FORMAT enumeration (dxgiformat.h)

Article 08/19/2022

Resource data formats, including fully-typed and typeless formats. A list of modifiers at the bottom of the page more fully describes each format type.

Syntax

C++

```
typedef enum DXGI_FORMAT {
    DXGI_FORMAT_UNKNOWN = 0,
    DXGI_FORMAT_R32G32B32A32_TYPELESS = 1,
    DXGI_FORMAT_R32G32B32A32_FLOAT = 2,
    DXGI_FORMAT_R32G32B32A32_UINT = 3,
    DXGI_FORMAT_R32G32B32A32_SINT = 4,
    DXGI_FORMAT_R32G32B32_TYPELESS = 5,
    DXGI_FORMAT_R32G32B32_FLOAT = 6,
    DXGI_FORMAT_R32G32B32_UINT = 7,
    DXGI_FORMAT_R32G32B32_SINT = 8,
    DXGI_FORMAT_R16G16B16A16_TYPELESS = 9,
    DXGI_FORMAT_R16G16B16A16_FLOAT = 10,
    DXGI_FORMAT_R16G16B16A16_UNORM = 11,
    DXGI_FORMAT_R16G16B16A16_UINT = 12,
    DXGI_FORMAT_R16G16B16A16_SNORM = 13,
    DXGI_FORMAT_R16G16B16A16_SINT = 14,
    DXGI_FORMAT_R32G32_TYPELESS = 15,
    DXGI_FORMAT_R32G32_FLOAT = 16,
    DXGI_FORMAT_R32G32_UINT = 17,
    DXGI_FORMAT_R32G32_SINT = 18,
    DXGI_FORMAT_R32G8X24_TYPELESS = 19,
    DXGI_FORMAT_D32_FLOAT_S8X24_UINT = 20,
    DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS = 21,
    DXGI_FORMAT_X32_TYPELESS_G8X24_UINT = 22,
    DXGI_FORMAT_R10G10B10A2_TYPELESS = 23,
    DXGI_FORMAT_R10G10B10A2_UNORM = 24,
    DXGI_FORMAT_R10G10B10A2_UINT = 25,
    DXGI_FORMAT_R11G11B10_FLOAT = 26,
    DXGI_FORMAT_R8G8B8A8_TYPELESS = 27,
    DXGI_FORMAT_R8G8B8A8_UNORM = 28,
    DXGI_FORMAT_R8G8B8A8_UNORM_SRGB = 29,
    DXGI_FORMAT_R8G8B8A8_UINT = 30,
    DXGI_FORMAT_R8G8B8A8_SNORM = 31,
    DXGI_FORMAT_R8G8B8A8_SINT = 32,
    DXGI_FORMAT_R16G16_TYPELESS = 33,
    DXGI_FORMAT_R16G16_FLOAT = 34,
    DXGI_FORMAT_R16G16_UNORM = 35,
```

```
DXGI_FORMAT_R16G16_UINT = 36,
DXGI_FORMAT_R16G16_SNORM = 37,
DXGI_FORMAT_R16G16_SINT = 38,
DXGI_FORMAT_R32_TYPELESS = 39,
DXGI_FORMAT_D32_FLOAT = 40,
DXGI_FORMAT_R32_FLOAT = 41,
DXGI_FORMAT_R32_UINT = 42,
DXGI_FORMAT_R32_SINT = 43,
DXGI_FORMAT_R24G8_TYPELESS = 44,
DXGI_FORMAT_D24_UNORM_S8_UINT = 45,
DXGI_FORMAT_R24_UNORM_X8_TYPELESS = 46,
DXGI_FORMAT_X24_TYPELESS_G8_UINT = 47,
DXGI_FORMAT_R8G8_TYPELESS = 48,
DXGI_FORMAT_R8G8_UNORM = 49,
DXGI_FORMAT_R8G8_UINT = 50,
DXGI_FORMAT_R8G8_SNORM = 51,
DXGI_FORMAT_R8G8_SINT = 52,
DXGI_FORMAT_R16_TYPELESS = 53,
DXGI_FORMAT_R16_FLOAT = 54,
DXGI_FORMAT_D16_UNORM = 55,
DXGI_FORMAT_R16_UNORM = 56,
DXGI_FORMAT_R16_UINT = 57,
DXGI_FORMAT_R16_SNORM = 58,
DXGI_FORMAT_R16_SINT = 59,
DXGI_FORMAT_R8_TYPELESS = 60,
DXGI_FORMAT_R8_UNORM = 61,
DXGI_FORMAT_R8_UINT = 62,
DXGI_FORMAT_R8_SNORM = 63,
DXGI_FORMAT_R8_SINT = 64,
DXGI_FORMAT_A8_UNORM = 65,
DXGI_FORMAT_R1_UNORM = 66,
DXGI_FORMAT_R9G9B9E5_SHAREDEXP = 67,
DXGI_FORMAT_R8G8_B8G8_UNORM = 68,
DXGI_FORMAT_G8R8_G8B8_UNORM = 69,
DXGI_FORMAT_BC1_TYPELESS = 70,
DXGI_FORMAT_BC1_UNORM = 71,
DXGI_FORMAT_BC1_UNORM_SRGB = 72,
DXGI_FORMAT_BC2_TYPELESS = 73,
DXGI_FORMAT_BC2_UNORM = 74,
DXGI_FORMAT_BC2_UNORM_SRGB = 75,
DXGI_FORMAT_BC3_TYPELESS = 76,
DXGI_FORMAT_BC3_UNORM = 77,
DXGI_FORMAT_BC3_UNORM_SRGB = 78,
DXGI_FORMAT_BC4_TYPELESS = 79,
DXGI_FORMAT_BC4_UNORM = 80,
DXGI_FORMAT_BC4_SNORM = 81,
DXGI_FORMAT_BC5_TYPELESS = 82,
DXGI_FORMAT_BC5_UNORM = 83,
DXGI_FORMAT_BC5_SNORM = 84,
DXGI_FORMAT_B5G6R5_UNORM = 85,
DXGI_FORMAT_B5G5R5A1_UNORM = 86,
DXGI_FORMAT_B8G8R8A8_UNORM = 87,
DXGI_FORMAT_B8G8R8X8_UNORM = 88,
DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM = 89,
DXGI_FORMAT_B8G8R8A8_TYPELESS = 90,
```

```
DXGI_FORMAT_B8G8R8A8_UNORM_SRGB = 91,
DXGI_FORMAT_B8G8R8X8_TYPELESS = 92,
DXGI_FORMAT_B8G8R8X8_UNORM_SRGB = 93,
DXGI_FORMAT_BC6H_TYPELESS = 94,
DXGI_FORMAT_BC6H_UF16 = 95,
DXGI_FORMAT_BC6H_SF16 = 96,
DXGI_FORMAT_BC7_TYPELESS = 97,
DXGI_FORMAT_BC7_UNORM = 98,
DXGI_FORMAT_BC7_UNORM_SRGB = 99,
DXGI_FORMAT_AYUV = 100,
DXGI_FORMAT_Y410 = 101,
DXGI_FORMAT_Y416 = 102,
DXGI_FORMAT_NV12 = 103,
DXGI_FORMAT_P010 = 104,
DXGI_FORMAT_P016 = 105,
DXGI_FORMAT_420_OPAQUE = 106,
DXGI_FORMAT_YUY2 = 107,
DXGI_FORMAT_Y210 = 108,
DXGI_FORMAT_Y216 = 109,
DXGI_FORMAT_NV11 = 110,
DXGI_FORMAT_AI44 = 111,
DXGI_FORMAT_IA44 = 112,
DXGI_FORMAT_P8 = 113,
DXGI_FORMAT_A8P8 = 114,
DXGI_FORMAT_B4G4R4A4_UNORM = 115,
DXGI_FORMAT_P208 = 130,
DXGI_FORMAT_V208 = 131,
DXGI_FORMAT_V408 = 132,
DXGI_FORMAT_SAMPLER_FEEDBACK_MIN_MIP_OPAQUE,
DXGI_FORMAT_SAMPLER_FEEDBACK_MIP_REGION_USED_OPAQUE,
DXGI_FORMAT_FORCE_UINT = 0xffffffff
} ;
```

Constants

`DXGI_FORMAT_UNKNOWN`

Value: 0

The format is not known.

`DXGI_FORMAT_R32G32B32A32_TYPELESS`

Value: 1

A four-component, 128-bit typeless format that supports 32 bits per channel including alpha.¹

`DXGI_FORMAT_R32G32B32A32_FLOAT`

Value: 2

A four-component, 128-bit floating-point format that supports 32 bits per channel including alpha.^{1,5,8}

`DXGI_FORMAT_R32G32B32A32_UINT`

Value: 3

A four-component, 128-bit unsigned-integer format that supports 32 bits per channel including alpha.¹

`DXGI_FORMAT_R32G32B32A32_SINT`

Value: 4

A four-component, 128-bit signed-integer format that supports 32 bits per channel including alpha.¹

`DXGI_FORMAT_R32G32B32_TYPELESS`

Value: 5

A three-component, 96-bit typeless format that supports 32 bits per color channel.

`DXGI_FORMAT_R32G32B32_FLOAT`

Value: 6

A three-component, 96-bit floating-point format that supports 32 bits per color channel.^{5,8}

`DXGI_FORMAT_R32G32B32_UINT`

Value: 7

A three-component, 96-bit unsigned-integer format that supports 32 bits per color channel.

`DXGI_FORMAT_R32G32B32_SINT`

Value: 8

A three-component, 96-bit signed-integer format that supports 32 bits per color channel.

`DXGI_FORMAT_R16G16B16A16_TYPELESS`

Value: 9

A four-component, 64-bit typeless format that supports 16 bits per channel including alpha.

`DXGI_FORMAT_R16G16B16A16_FLOAT`

Value: 10

A four-component, 64-bit floating-point format that supports 16 bits per channel including alpha.^{5,7}

`DXGI_FORMAT_R16G16B16A16_UNORM`

Value: 11

A four-component, 64-bit unsigned-normalized-integer format that supports 16 bits per channel including alpha.

`DXGI_FORMAT_R16G16B16A16_UINT`

Value: 12

A four-component, 64-bit unsigned-integer format that supports 16 bits per channel including alpha.

`DXGI_FORMAT_R16G16B16A16_SNORM`

Value: 13

A four-component, 64-bit signed-normalized-integer format that supports 16 bits per channel including alpha.

`DXGI_FORMAT_R16G16B16A16_SINT`

Value: 14

A four-component, 64-bit signed-integer format that supports 16 bits per channel including alpha.

`DXGI_FORMAT_R32G32_TYPELESS`

Value: 15

A two-component, 64-bit typeless format that supports 32 bits for the red channel and 32 bits for the green channel.

`DXGI_FORMAT_R32G32_FLOAT`

Value: 16

A two-component, 64-bit floating-point format that supports 32 bits for the red channel and 32 bits for the green channel.^{5,8}

`DXGI_FORMAT_R32G32_UINT`

Value: 17

A two-component, 64-bit unsigned-integer format that supports 32 bits for the red channel and 32 bits for the green channel.

`DXGI_FORMAT_R32G32_SINT`

Value: 18

A two-component, 64-bit signed-integer format that supports 32 bits for the red channel and 32 bits for the green channel.

`DXGI_FORMAT_R32G8X24_TYPELESS`

Value: 19

A two-component, 64-bit typeless format that supports 32 bits for the red channel, 8 bits for the green channel, and 24 bits are unused.

`DXGI_FORMAT_D32_FLOAT_S8X24_UINT`

Value: 20

A 32-bit floating-point component, and two unsigned-integer components (with an additional 32 bits). This format supports 32-bit depth, 8-bit stencil, and 24 bits are unused.⁵

`DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS`

Value: 21

A 32-bit floating-point component, and two typeless components (with an additional 32 bits). This format supports 32-bit red channel, 8 bits are unused, and 24 bits are unused.⁵

`DXGI_FORMAT_X32_TYPELESS_G8X24_UINT`

Value: 22

A 32-bit typeless component, and two unsigned-integer components (with an additional 32 bits). This format has 32 bits unused, 8 bits for green channel, and 24 bits are unused.

`DXGI_FORMAT_R10G10B10A2_TYPELESS`

Value: 23

A four-component, 32-bit typeless format that supports 10 bits for each color and 2 bits for alpha.

`DXGI_FORMAT_R10G10B10A2_UNORM`

Value: 24

A four-component, 32-bit unsigned-normalized-integer format that supports 10 bits for each color and 2 bits for alpha.

`DXGI_FORMAT_R10G10B10A2_UINT`

Value: 25

A four-component, 32-bit unsigned-integer format that supports 10 bits for each color and 2 bits for alpha.

`DXGI_FORMAT_R11G11B10_FLOAT`

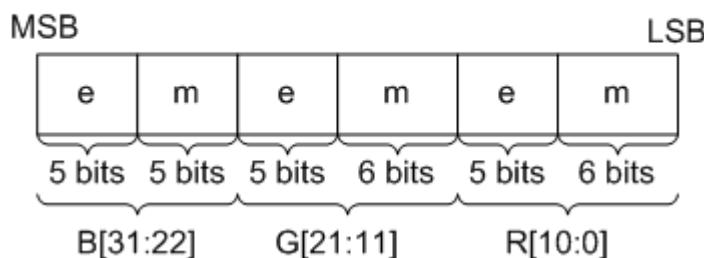
Value: 26

Three partial-precision floating-point numbers encoded into a single 32-bit value (a variant of s10e5, which is sign bit, 10-bit mantissa, and 5-bit biased (15) exponent).

There are no sign bits, and there is a 5-bit biased (15) exponent for each channel, 6-bit mantissa for R and G, and a 5-bit mantissa for B, as shown in the following illustration.^{5,7}

32 Bits Per Element:

e = exponent
m = mantissa



`DXGI_FORMAT_R8G8B8A8_TYPELESS`

Value: 27

A four-component, 32-bit typeless format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R8G8B8A8_UNORM`

Value: 28

A four-component, 32-bit unsigned-normalized-integer format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R8G8B8A8_UNORM_SRGB`

Value: 29

A four-component, 32-bit unsigned-normalized integer sRGB format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R8G8B8A8_UINT`

Value: 30

A four-component, 32-bit unsigned-integer format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R8G8B8A8_SNORM`

Value: 31

A four-component, 32-bit signed-normalized-integer format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R8G8B8A8_SINT`

Value: 32

A four-component, 32-bit signed-integer format that supports 8 bits per channel including alpha.

`DXGI_FORMAT_R16G16_TYPELESS`

Value: 33

A two-component, 32-bit typeless format that supports 16 bits for the red channel and 16 bits for the green channel.

`DXGI_FORMAT_R16G16_FLOAT`

Value: 34

A two-component, 32-bit floating-point format that supports 16 bits for the red channel and 16 bits for the green channel.^{5,7}

`DXGI_FORMAT_R16G16_UNORM`

Value: 35

A two-component, 32-bit unsigned-normalized-integer format that supports 16 bits each for the green and red channels.

`DXGI_FORMAT_R16G16_UINT`

Value: 36

A two-component, 32-bit unsigned-integer format that supports 16 bits for the red channel and 16 bits for the green channel.

`DXGI_FORMAT_R16G16_SNORM`

Value: 37

A two-component, 32-bit signed-normalized-integer format that supports 16 bits for the red channel and 16 bits for the green channel.

`DXGI_FORMAT_R16G16_SINT`

Value: 38

A two-component, 32-bit signed-integer format that supports 16 bits for the red channel and 16 bits for the green channel.

`DXGI_FORMAT_R32_TYPELESS`

Value: 39

A single-component, 32-bit typeless format that supports 32 bits for the red channel.

`DXGI_FORMAT_D32_FLOAT`

Value: 40

A single-component, 32-bit floating-point format that supports 32 bits for depth.^{5,8}

`DXGI_FORMAT_R32_FLOAT`

Value: 41

A single-component, 32-bit floating-point format that supports 32 bits for the red channel.^{5,8}

`DXGI_FORMAT_R32_UINT`

Value: 42

A single-component, 32-bit unsigned-integer format that supports 32 bits for the red channel.

`DXGI_FORMAT_R32_SINT`

Value: 43

A single-component, 32-bit signed-integer format that supports 32 bits for the red channel.

`DXGI_FORMAT_R24G8_TYPELESS`

Value: 44

A two-component, 32-bit typeless format that supports 24 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_D24_UNORM_S8_UINT`

Value: 45

A 32-bit z-buffer format that supports 24 bits for depth and 8 bits for stencil.

`DXGI_FORMAT_R24_UNORM_X8_TYPELESS`

Value: 46

A 32-bit format, that contains a 24 bit, single-component, unsigned-normalized integer, with an additional typeless 8 bits. This format has 24 bits red channel and 8 bits unused.

`DXGI_FORMAT_X24_TYPELESS_G8_UINT`

Value: 47

A 32-bit format, that contains a 24 bit, single-component, typeless format, with an additional 8 bit unsigned integer component. This format has 24 bits unused and 8 bits green channel.

`DXGI_FORMAT_R8G8_TYPELESS`

Value: 48

A two-component, 16-bit typeless format that supports 8 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_R8G8_UNORM`

Value: 49

A two-component, 16-bit unsigned-normalized-integer format that supports 8 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_R8G8_UINT`

Value: 50

A two-component, 16-bit unsigned-integer format that supports 8 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_R8G8_SNORM`

Value: 51

A two-component, 16-bit signed-normalized-integer format that supports 8 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_R8G8_SINT`

Value: 52

A two-component, 16-bit signed-integer format that supports 8 bits for the red channel and 8 bits for the green channel.

`DXGI_FORMAT_R16_TYPELESS`

Value: 53

A single-component, 16-bit typeless format that supports 16 bits for the red channel.

`DXGI_FORMAT_R16_FLOAT`

Value: 54

A single-component, 16-bit floating-point format that supports 16 bits for the red channel.^{5,7}

`DXGI_FORMAT_D16_UNORM`

Value: 55

A single-component, 16-bit unsigned-normalized-integer format that supports 16 bits for depth.

`DXGI_FORMAT_R16_UNORM`

Value: 56

A single-component, 16-bit unsigned-normalized-integer format that supports 16 bits for the red channel.

`DXGI_FORMAT_R16_UINT`

Value: 57

A single-component, 16-bit unsigned-integer format that supports 16 bits for the red channel.

`DXGI_FORMAT_R16_SNORM`

Value: 58

A single-component, 16-bit signed-normalized-integer format that supports 16 bits for the red channel.

`DXGI_FORMAT_R16_SINT`

Value: 59

A single-component, 16-bit signed-integer format that supports 16 bits for the red channel.

`DXGI_FORMAT_R8_TYPELESS`

Value: 60

A single-component, 8-bit typeless format that supports 8 bits for the red channel.

DXGI_FORMAT_R8_UNORM

Value: 61

A single-component, 8-bit unsigned-normalized-integer format that supports 8 bits for the red channel.

DXGI_FORMAT_R8_UINT

Value: 62

A single-component, 8-bit unsigned-integer format that supports 8 bits for the red channel.

DXGI FORMAT R8 SNORM

Value: 63

A single-component, 8-bit signed-normalized-integer format that supports 8 bits for the red channel.

DXGI FORMAT R8 SINT

Value: 64

A single-component, 8-bit signed-integer format that supports 8 bits for the red channel.

DXGI FORMAT A8 UNORM

Value: 65

A single-component, 8-bit unsigned-normalized-integer format for alpha only.

DXGI FORMAT R1 UNORM

Value: 66

A single-component, 1-bit unsigned-normalized integer format that supports 1 bit for the red channel.²

DXGI FORMAT R9G9B9E5 SHAREDEXP

Value: 67

Three partial-precision floating-point numbers encoded into a single 32-bit value all sharing the same 5-bit exponent (variant of s10e5, which is sign bit, 10-bit mantissa, and 5-bit biased (15) exponent).

There is no sign bit, and there is a shared 5-bit biased (15) exponent and a 9-bit mantissa for each channel, as shown in the following illustration.^{6,7}

32 Bits Per Element:

MSB

158



DXGI FORMAT R8G8 B8G8 UNORM

Value: 68

A four-component, 32-bit unsigned-normalized-integer format. This packed RGB format is analogous to the UYVY format. Each 32-bit block describes a pair of pixels: (R8, G8, B8) and (R8, G8, B8) where the R8/B8 values are repeated, and the G8 values are unique to each pixel.³

Width must be even.

`DXGI_FORMAT_G8R8_G8B8_UNORM`

Value: 69

A four-component, 32-bit unsigned-normalized-integer format. This packed RGB format is analogous to the YUY2 format. Each 32-bit block describes a pair of pixels: (R8, G8, B8) and (R8, G8, B8) where the R8/B8 values are repeated, and the G8 values are unique to each pixel.³

Width must be even.

`DXGI_FORMAT_BC1_TYPELESS`

Value: 70

Four-component typeless block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC1_UNORM`

Value: 71

Four-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC1_UNORM_SRGB`

Value: 72

Four-component block-compression format for sRGB data. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC2_TYPELESS`

Value: 73

Four-component typeless block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC2_UNORM`

Value: 74

Four-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC2_UNORM_SRGB`

Value: 75

Four-component block-compression format for sRGB data. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC3_TYPELESS`

Value: 76

Four-component typeless block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC3_UNORM`

Value: 77

Four-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC3_UNORM_SRGB`

Value: 78

Four-component block-compression format for sRGB data. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC4_TYPELESS`

Value: 79

One-component typeless block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC4_UNORM`

Value: 80

One-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC4_SNORM`

Value: 81

One-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC5_TYPELESS`

Value: 82

Two-component typeless block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC5_UNORM`

Value: 83

Two-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC5_SNORM`

Value: 84

Two-component block-compression format. For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_B5G6R5_UNORM`

Value: 85

A three-component, 16-bit unsigned-normalized-integer format that supports 5 bits for blue, 6 bits for green, and 5 bits for red.

Direct3D 10 through Direct3D 11: This value is defined for DXGI. However, Direct3D 10, 10.1, or 11 devices do not support this format.

Direct3D 11.1: This value is not supported until Windows 8.

`DXGI_FORMAT_B5G5R5A1_UNORM`

Value: 86

A four-component, 16-bit unsigned-normalized-integer format that supports 5 bits for each color channel and 1-bit alpha.

Direct3D 10 through Direct3D 11: This value is defined for DXGI. However, Direct3D 10, 10.1, or 11 devices do not support this format.

Direct3D 11.1: This value is not supported until Windows 8.

`DXGI_FORMAT_B8G8R8A8_UNORM`

Value: 87

A four-component, 32-bit unsigned-normalized-integer format that supports 8 bits for each color channel and 8-bit alpha.

`DXGI_FORMAT_B8G8R8X8_UNORM`

Value: 88

A four-component, 32-bit unsigned-normalized-integer format that supports 8 bits for each color channel and 8 bits unused.

`DXGI_FORMAT_R10G10B10_XR_BIAS_A2_UNORM`

Value: 89

A four-component, 32-bit 2.8-biased fixed-point format that supports 10 bits for each color channel and 2-bit alpha.

`DXGI_FORMAT_B8G8R8A8_TYPELESS`

Value: 90

A four-component, 32-bit typeless format that supports 8 bits for each channel including alpha.⁴

`DXGI_FORMAT_B8G8R8A8_UNORM_SRGB`

Value: 91

A four-component, 32-bit unsigned-normalized standard RGB format that supports 8 bits for each channel including alpha.⁴

`DXGI_FORMAT_B8G8R8X8_TYPELESS`

Value: 92

A four-component, 32-bit typeless format that supports 8 bits for each color channel, and 8 bits are unused.⁴

`DXGI_FORMAT_B8G8R8X8_UNORM_SRGB`

Value: 93

A four-component, 32-bit unsigned-normalized standard RGB format that supports 8 bits for each color channel, and 8 bits are unused.⁴

`DXGI_FORMAT_BC6H_TYPELESS`

Value: 94

A typeless block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC6H_UF16`

Value: 95

A block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).⁵

`DXGI_FORMAT_BC6H_SF16`

Value: 96

A block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).⁵

`DXGI_FORMAT_BC7_TYPELESS`

Value: 97

A typeless block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC7_UNORM`

Value: 98

A block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_BC7_UNORM_SRGB`

Value: 99

A block-compression format.⁴ For information about block-compression formats, see [Texture Block Compression in Direct3D 11](#).

`DXGI_FORMAT_AYUV`

Value: 100

Most common YUV 4:4:4 video resource format. Valid view formats for this video resource format are `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UINT`. For UAVs, an additional valid view format is `DXGI_FORMAT_R32_UINT`. By using `DXGI_FORMAT_R32_UINT` for UAVs, you can both read and write as opposed to just write for `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UINT`. Supported view types are SRV, RTV, and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is V->R8, U->G8, Y->B8, and A->A8.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_Y410

Value: 101

10-bit per channel packed YUV 4:4:4 video resource format. Valid view formats for this video resource format are DXGI_FORMAT_R10G10B10A2_UNORM and DXGI_FORMAT_R10G10B10A2_UINT. For UAVs, an additional valid view format is DXGI_FORMAT_R32_UINT. By using DXGI_FORMAT_R32_UINT for UAVs, you can both read and write as opposed to just write for DXGI_FORMAT_R10G10B10A2_UNORM and DXGI_FORMAT_R10G10B10A2_UINT. Supported view types are SRV and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is U->R10, Y->G10, V->B10, and A->A2.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_Y416

Value: 102

16-bit per channel packed YUV 4:4:4 video resource format. Valid view formats for this video resource format are DXGI_FORMAT_R16G16B16A16_UNORM and DXGI_FORMAT_R16G16B16A16_UINT. Supported view types are SRV and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is U->R16, Y->G16, V->B16, and A->A16.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

`DXGI_FORMAT_NV12`

Value: 103

Most common YUV 4:2:0 video resource format. Valid luminance data view formats for this video resource format are `DXGI_FORMAT_R8_UNORM` and `DXGI_FORMAT_R8_UINT`. Valid chrominance data view formats (width and height are each 1/2 of luminance view) for this video resource format are `DXGI_FORMAT_R8G8_UNORM` and `DXGI_FORMAT_R8G8_UINT`. Supported view types are SRV, RTV, and UAV. For luminance data view, the mapping to the view channel is Y->R8. For chrominance data view, the mapping to the view channel is U->R8 and V->G8.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width and height must be even. Direct3D 11 staging resources and initData parameters for this format use $(\text{rowPitch} * (\text{height} + (\text{height} / 2)))$ bytes. The first $(\text{SysMemPitch} * \text{height})$ bytes are the Y plane, the remaining $(\text{SysMemPitch} * (\text{height} / 2))$ bytes are the UV plane.

An app using the YUY 4:2:0 formats must map the luma (Y) plane separately from the chroma (UV) planes. Developers do this by calling [`ID3D12Device::CreateShaderResourceView`](#) twice for the same texture and passing in 1-channel and 2-channel formats. Passing in a 1-channel format compatible with the Y plane maps only the Y plane. Passing in a 2-channel format compatible with the UV planes (together) maps only the U and V planes as a single resource view.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_P010

Value: 104

10-bit per channel planar YUV 4:2:0 video resource format. Valid luminance data view formats for this video resource format are DXGI_FORMAT_R16_UNORM and DXGI_FORMAT_R16_UINT. The runtime does not enforce whether the lowest 6 bits are 0 (given that this video resource format is a 10-bit format that uses 16 bits). If required, application shader code would have to enforce this manually. From the runtime's point of view, DXGI_FORMAT_P010 is no different than DXGI_FORMAT_P016. Valid chrominance data view formats (width and height are each 1/2 of luminance view) for this video resource format are DXGI_FORMAT_R16G16_UNORM and DXGI_FORMAT_R16G16_UINT. For UAVs, an additional valid chrominance data view format is DXGI_FORMAT_R32_UINT. By using DXGI_FORMAT_R32_UINT for UAVs, you can both read and write as opposed to just write for DXGI_FORMAT_R16G16_UNORM and DXGI_FORMAT_R16G16_UINT. Supported view types are SRV, RTV, and UAV. For luminance data view, the mapping to the view channel is Y->R16. For chrominance data view, the mapping to the view channel is U->R16 and V->G16.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width and height must be even. Direct3D 11 staging resources and initData parameters for this format use $(\text{rowPitch} * (\text{height} + (\text{height} / 2)))$ bytes. The first $(\text{SysMemPitch} * \text{height})$ bytes are the Y plane, the remaining $(\text{SysMemPitch} * (\text{height} / 2))$ bytes are the UV plane.

An app using the YUY 4:2:0 formats must map the luma (Y) plane separately from the chroma (UV) planes. Developers do this by calling [ID3D12Device::CreateShaderResourceView](#) twice for the same texture and passing in 1-channel and 2-channel formats. Passing in a 1-channel format compatible with the Y plane maps only the Y plane. Passing in a 2-channel format compatible with the UV planes (together) maps only the U and V planes as a single resource view.

Direct3D 11.1: This value is not supported until Windows 8.

`DXGI_FORMAT_P016`

Value: 105

16-bit per channel planar YUV 4:2:0 video resource format. Valid luminance data view formats for this video resource format are `DXGI_FORMAT_R16_UNORM` and `DXGI_FORMAT_R16_UINT`. Valid chrominance data view formats (width and height are each 1/2 of luminance view) for this video resource format are `DXGI_FORMAT_R16G16_UNORM` and `DXGI_FORMAT_R16G16_UINT`. For UAVs, an additional valid chrominance data view format is `DXGI_FORMAT_R32_UINT`. By using `DXGI_FORMAT_R32_UINT` for UAVs, you can both read and write as opposed to just write for `DXGI_FORMAT_R16G16_UNORM` and `DXGI_FORMAT_R16G16_UINT`. Supported view types are SRV, RTV, and UAV. For luminance data view, the mapping to the view channel is Y->R16. For chrominance data view, the mapping to the view channel is U->R16 and V->G16.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width and height must be even. Direct3D 11 staging resources and initData parameters for this format use $(\text{rowPitch} * (\text{height} + (\text{height} / 2)))$ bytes. The first $(\text{SysMemPitch} * \text{height})$ bytes are the Y plane, the remaining $(\text{SysMemPitch} * (\text{height} / 2))$ bytes are the UV plane.

An app using the YUY 4:2:0 formats must map the luma (Y) plane separately from the chroma (UV) planes. Developers do this by calling [`ID3D12Device::CreateShaderResourceView`](#) twice for the same texture and passing in 1-channel and 2-channel formats. Passing in a 1-channel format compatible with the Y plane maps only the Y plane. Passing in a 2-channel format compatible with the UV planes (together) maps only the U and V planes as a single resource view.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_420_OPAQUE

Value: 106

8-bit per channel planar YUV 4:2:0 video resource format. This format is subsampled where each pixel has its own Y value, but each 2x2 pixel block shares a single U and V value. The runtime requires that the width and height of all resources that are created with this format are multiples of 2. The runtime also requires that the left, right, top, and bottom members of any RECT that are used for this format are multiples of 2. This format differs from DXGI_FORMAT_NV12 in that the layout of the data within the resource is completely opaque to applications. Applications cannot use the CPU to map the resource and then access the data within the resource. You cannot use shaders with this format. Because of this behavior, legacy hardware that supports a non-NV12 4:2:0 layout (for example, YV12, and so on) can be used. Also, new hardware that has a 4:2:0 implementation better than NV12 can be used when the application does not need the data to be in a standard layout.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width and height must be even. Direct3D 11 staging resources and initData parameters for this format use (rowPitch * (height + (height / 2))) bytes.

An app using the YUY 4:2:0 formats must map the luma (Y) plane separately from the chroma (UV) planes. Developers do this by calling [ID3D12Device::CreateShaderResourceView](#) twice for the same texture and passing in 1-channel and 2-channel formats. Passing in a 1-channel format compatible with the Y plane maps only the Y plane. Passing in a 2-channel format compatible with the UV planes (together) maps only the U and V planes as a single resource view.

Direct3D 11.1: This value is not supported until Windows 8.

`DXGI_FORMAT_YUY2`

Value: 107

Most common YUV 4:2:2 video resource format. Valid view formats for this video resource format are `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UINT`. For UAVs, an additional valid view format is `DXGI_FORMAT_R32_UINT`. By using `DXGI_FORMAT_R32_UINT` for UAVs, you can both read and write as opposed to just write for `DXGI_FORMAT_R8G8B8A8_UNORM` and `DXGI_FORMAT_R8G8B8A8_UINT`. Supported view types are SRV and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is Y0->R8,
U0->G8,
Y1->B8,
and V0->A8.

A unique valid view format for this video resource format is `DXGI_FORMAT_R8G8_B8G8_UNORM`. With this view format, the width of the view appears to be twice what the `DXGI_FORMAT_R8G8B8A8_UNORM` or `DXGI_FORMAT_R8G8B8A8_UINT` view would be when hardware reconstructs RGBA automatically on read and before filtering. This Direct3D hardware behavior is legacy and is likely not useful any more. With this view format, the mapping to the view channel is Y0->R8,
U0->
G8[0],
Y1->B8,
and V0->
G8[1].

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width must be even.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_Y210

Value: 108

10-bit per channel packed YUV 4:2:2 video resource format. Valid view formats for this video resource format are DXGI_FORMAT_R16G16B16A16_UNORM and DXGI_FORMAT_R16G16B16A16_UINT. The runtime does not enforce whether the lowest 6 bits are 0 (given that this video resource format is a 10-bit format that uses 16 bits). If required, application shader code would have to enforce this manually. From the runtime's point of view, DXGI_FORMAT_Y210 is no different than DXGI_FORMAT_Y216. Supported view types are SRV and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is Y0->R16, U->G16, Y1->B16, and V->A16.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width must be even.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_Y216

Value: 109

16-bit per channel packed YUV 4:2:2 video resource format. Valid view formats for this video resource format are DXGI_FORMAT_R16G16B16A16_UNORM and DXGI_FORMAT_R16G16B16A16_UINT. Supported view types are SRV and UAV. One view provides a straightforward mapping of the entire surface. The mapping to the view channel is Y0->R16, U->G16, Y1->B16, and V->A16.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width must be even.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_NV11

Value: 110

Most common planar YUV 4:1:1 video resource format. Valid luminance data view formats for this video resource format are DXGI_FORMAT_R8_UNORM and DXGI_FORMAT_R8_UINT. Valid chrominance data view formats (width and height are each 1/4 of luminance view) for this video resource format are DXGI_FORMAT_R8G8_UNORM and DXGI_FORMAT_R8G8_UINT. Supported view types are SRV, RTV, and UAV. For luminance data view, the mapping to the view channel is Y->R8. For chrominance data view, the mapping to the view channel is U->R8 and V->G8.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Width must be a multiple of 4. Direct3D11 staging resources and initData parameters for this format use $(\text{rowPitch} * \text{height} * 2)$ bytes. The first $(\text{SysMemPitch} * \text{height})$ bytes are the Y plane, the next $((\text{SysMemPitch} / 2) * \text{height})$ bytes are the UV plane, and the remainder is padding.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_AI44

Value: 111

4-bit palletized YUV format that is commonly used for DVD subpicture.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_IA44

Value: 112

4-bit palletized YUV format that is commonly used for DVD subpicture.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_P8

Value: 113

8-bit palletized format that is used for palletized RGB data when the processor processes ISDB-T data and for palletized YUV data when the processor processes BluRay data.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_A8P8

Value: 114

8-bit palletized format with 8 bits of alpha that is used for palletized YUV data when the processor processes BluRay data.

For more info about YUV formats for video rendering, see [Recommended 8-Bit YUV Formats for Video Rendering](#).

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_B4G4R4A4_UNORM

Value: 115

A four-component, 16-bit unsigned-normalized integer format that supports 4 bits for each channel including alpha.

Direct3D 11.1: This value is not supported until Windows 8.

DXGI_FORMAT_P208

Value: 130

A video format; an 8-bit version of a hybrid planar 4:2:2 format.

DXGI_FORMAT_V208

Value: 131

An 8 bit YCbCrA 4:4 rendering format.

DXGI_FORMAT_V408

Value: 132

An 8 bit YCbCrA 4:4:4 rendering format.

DXGI_FORMAT_FORCE_UINT

Value: 0xffffffff

Forces this enumeration to compile to 32 bits in size. Without this value, some compilers would allow this enumeration to compile to a size other than 32 bits. This value is not used.

Remarks

Byte Order (LSB/MSB)

Most formats have byte-aligned components, and the components are in C-array order (the least address comes first). For those formats that don't have power-of-2-aligned components, the first named component is in the least-significant bits.

Portable Coding for Endian-Independence

Rather than adjusting for whether a system uses big-endian or little-endian byte ordering, you should write portable code, as follows.

```
// DXGI_FORMAT_R32G32B32A32_FLOAT
FLOAT* pR32G32B32A32 = ...;
pR32G32B32A32[0] = 1.f; // R
pR32G32B32A32[1] = 0.f; // G
pR32G32B32A32[2] = 0.f; // B
pR32G32B32A32[3] = 0.5f; // A

// DXGI_FORMAT_R10G10B10A2_UNORM
UINT32* pR10G10B10A2 = ...;
pR10G10B10A2 = (0x3ff) | (0x1 << 30); // R=0x3ff, and A=0x1
```

Restrictions and notes on formats

A few formats have additional restrictions and implied behavior:

1. A resource declared with the DXGI_FORMAT_R32G32B32 family of formats cannot be used simultaneously for vertex and texture data. That is, you may not create a buffer resource with the DXGI_FORMAT_R32G32B32 family of formats that uses any of the following bind flags: D3D10_BIND_VERTEX_BUFFER, D3D10_BIND_INDEX_BUFFER, D3D10_BIND_CONSTANT_BUFFER, or D3D10_BIND_STREAM_OUTPUT (see [D3D10_BIND_FLAG](#)).
2. DXGI_FORMAT_R1_UNORM is designed specifically for text filtering, and must be used with a format-specific, configurable 8x8 filter mode. When calling an HLSL sampling function using this format, the address offset parameter must be set to (0,0).
3. A resource using a sub-sampled format (such as DXGI_FORMAT_R8G8_B8G8) must have a size that is a multiple of 2 in the x dimension.
4. Format is not available in Direct3D 10 and Direct3D 10.1
5. These float formats have an implied 1 added to their mantissa. If the exponent is not 0, 1.0 is added to the mantissa before applying the exponent.
6. These float formats do not have an implied 1 added to their mantissa.
7. Denorm support: the 9, 10, 11 and 16 bit float formats support denorms.
8. No denorm support: the 32 and 64 bit float formats flush denorms to zero.

The following topics provide lists of the formats that particular hardware [feature levels](#) support:

- DXGI Format Support for Direct3D Feature Level 12.1 Hardware
- DXGI Format Support for Direct3D Feature Level 12.0 Hardware
- DXGI Format Support for Direct3D Feature Level 11.1 Hardware
- DXGI Format Support for Direct3D Feature Level 11.0 Hardware
- Hardware Support for Direct3D 10Level9 Formats
- Hardware Support for Direct3D 10.1 Formats
- Hardware Support for Direct3D 10 Formats

For a list of the **DirectXMath** types that map to **DXGI_FORMAT** values, see [DirectXMath Library Internals](#).

Format Modifiers

Each enumeration value contains a format modifier which describes the data type.

Format Modifiers	Description
_FLOAT	A floating-point value; 32-bit floating-point formats use IEEE 754 single-precision (s23e8 format): sign bit, 8-bit biased (127) exponent, and 23-bit mantissa. 16-bit floating-point formats use half-precision (s10e5 format): sign bit, 5-bit biased (15) exponent, and 10-bit mantissa.
_SINT	Two's complement signed integer. For example, a 3-bit SINT represents the values -4, -3, -2, -1, 0, 1, 2, 3.
_SNORM	Signed normalized integer; which is interpreted in a resource as a signed integer, and is interpreted in a shader as a signed normalized floating-point value in the range [-1, 1]. For a 2's complement number, the maximum value is 1.0f (a 5-bit value 01111 maps to 1.0f), and the minimum value is -1.0f (a 5-bit value 10000 maps to -1.0f). In addition, the second-minimum number maps to -1.0f (a 5-bit value 10001 maps to -1.0f). The resulting integer representations are evenly spaced floating-point values in the range (-1.0f...0.0f), and also a complementary set of representations for numbers in the range (0.0f...1.0f).
_SRGB	Standard RGB data, which roughly displays colors in a linear ramp of luminosity levels such that an average observer, under average viewing conditions, can view them on an average display. All 0's maps to 0.0f, and all 1's maps to 1.0f. The sequence of unsigned integer encodings between all 0's and all 1's represent a nonlinear progression in the floating-point interpretation of the numbers between 0.0f to 1.0f. For more detail, see the SRGB color standard, IEC 61996-2-1, at IEC (International Electrotechnical Commission). Conversion to or from sRGB space is automatically done by D3DX10 or D3DX9 texture-load functions. If a format with _SRGB has an A channel, the A channel is stored in Gamma 1.0f data; the R, G, and B channels in the format are stored in sRGB Gamma (linear segment + 2.4 power) data.

_TYPELESS	Typeless data, with a defined number of bits. Typeless formats are designed for creating typeless resources; that is, a resource whose size is known, but whose data type is not yet fully defined. When a typeless resource is bound to a shader, the application or shader must resolve the format type (which must match the number of bits per component in the typeless format). A typeless format contains one or more subformats; each subformat resolves the data type. For example, in the R32G32B32 group, which defines types for three-component 96-bit data, there is one typeless format and three fully typed subformats.
	<pre>DXGI_FORMAT_R32G32B32_TYPELESS, DXGI_FORMAT_R32G32B32_FLOAT, DXGI_FORMAT_R32G32B32_UINT, DXGI_FORMAT_R32G32B32_SINT,</pre>
_UINT	Unsigned integer. For instance, a 3-bit UINT represents the values 0, 1, 2, 3, 4, 5, 6, 7.
_UNORM	Unsigned normalized integer; which is interpreted in a resource as an unsigned integer, and is interpreted in a shader as an unsigned normalized floating-point value in the range [0, 1]. All 0's maps to 0.0f, and all 1's maps to 1.0f. A sequence of evenly spaced floating-point values from 0.0f to 1.0f are represented. For instance, a 2-bit UNORM represents 0.0f, 1/3, 2/3, and 1.0f.
_SHAREDEXP	A shared exponent. All the floating point representations in the format share the one exponent.

New Resource Formats

Direct3D 10 offers new data compression formats for compressing high-dynamic range (HDR) lighting data, normal maps and heightfields to a fraction of their original size. These compression types include:

- Shared-Exponent high-dynamic range (HDR) format (RGBE)
- New Block-Compressed 1-2 channel UNORM/SNORM formats

The block compression formats can be used for any of the 2D or 3D texture types (Texture2D, Texture2DArray, Texture3D, or TextureCube) including mipmap surfaces. The block compression techniques require texture dimensions to be a multiple of 4 (since the implementation compresses on blocks of 4x4 texels). In the texture sampler, compressed formats are always decompressed before texture filtering.

Requirements

Header	dxgiformat.h
--------	--------------

See also

[DXGI Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

winerror.h header

Article01/24/2023

This header is used by multiple technologies. For more information, see:

- [Component Object Model \(COM\)](#)
- [Direct2D](#)
- [DXGI](#)
- [Network Access Protection](#)
- [Remote Procedure Call \(RPC\)](#)
- [System Services](#)
- [Task Scheduler](#)
- [Windows Animation Manager](#)
- [Windows Event Log](#)
- [Windows Filtering Platform](#)

winerror.h contains the following programming interfaces:

Functions

FAILED
Provides a generic test for failure on any status value.
HRESULT_CODE
Extracts the code portion of the specified HRESULT.
HRESULT_FACILITY
Extracts the facility of the specified HRESULT.
HRESULT_FROM_NT
Maps an NT status value to an HRESULT value.
HRESULT_FROM_WIN32
Maps a system error code to an HRESULT value.
HRESULT_SEVERITY
Extracts the severity field of the specified HRESULT.

[IS_ERROR](#)

Provides a generic test for errors on any status value.

[MAKE_HRESULT](#)

The MAKE_HRESULT macro (winerror.h) creates an HRESULT value from its component pieces.

[MAKE_SCODE](#)

Creates an SCODE value from its component pieces.

[SCODE_CODE](#)

Extracts the code portion of the specified SCODE.

[SCODE_FACILITY](#)

Extracts the facility of the specified SCODE.

[SCODE_SEVERITY](#)

Extracts the severity field of the specified SCODE.

[SUCCEEDED](#)

Provides a generic test for success on any status value.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

winnt.h header

Article02/16/2023

This header is used by multiple technologies. For more information, see:

- [Application Installation and Servicing](#)
- [Backup](#)
- [Data Access and Storage](#)
- [Developer Notes](#)
- [DXGI](#)
- [Hardware Counter Profiling](#)
- [Internationalization for Windows Applications](#)
- [Kernel-Mode Driver Reference](#)
- [Menus and Other Resources](#)
- [Security and Identity](#)
- [System Services](#)
- [The Windows Shell](#)
- [Windows Management Instrumentation](#)
- [Windows Runtime C++ reference](#)

winnt.h contains the following programming interfaces:

Functions

[_InlinelInterlockedAdd](#)

Performs an atomic addition operation on the specified LONG values. The operation is performed with acquire memory ordering semantics.

[_InlinelInterlockedAdd64](#)

Performs an atomic addition operation on the specified LONG64 values.

[_interlockedbittestandreset](#)

Tests the specified bit of the specified LONG value and sets it to 0. The operation is atomic.

[_interlockedbittestandreset64](#)

Tests the specified bit of the specified LONG64 value and sets it to 0. The operation is atomic.

[_interlockedbittestandset](#)

Tests the specified bit of the specified LONG value and sets it to 1. The operation is atomic.

[_interlockedbittestandset64](#)

Tests the specified bit of the specified LONG64 value and sets it to 1. The operation is atomic.

[C_ASSERT](#)

Checks assertions at compile time.

[FIELD_OFFSET](#)

The FIELD_OFFSET macro returns the byte offset of a named field in a known structure type.
(FIELD_OFFSET macro)

[GetCurrentFiber](#)

Retrieves the address of the current fiber.

[GetFiberData](#)

Retrieves the fiber data associated with the current fiber.

[Int32x32To64](#)

Multiplies two signed 32-bit integers, returning a signed 64-bit integer result.

[Int64ShlMod32](#)

Performs a left logical shift operation on an unsigned 64-bit integer value. The function provides improved shifting code for left logical shifts where the shift count is in the range 0-31.

[Int64ShraMod32](#)

Performs a right arithmetic shift operation on a signed 64-bit integer value. The function provides improved shifting code for right arithmetic shifts where the shift count is in the range 0-31.

[Int64ShrlMod32](#)

Performs a right logical shift operation on an unsigned 64-bit integer value. The function provides improved shifting code for right logical shifts where the shift count is in the range 0-31.

[InterlockedAdd](#)

Performs an atomic addition operation on the specified LONG values.

InterlockedAnd	Performs an atomic AND operation on the specified LONG values.
InterlockedAnd16	Performs an atomic AND operation on the specified SHORT values.
InterlockedAnd64	Performs an atomic AND operation on the specified LONGLONG values.
InterlockedAnd8	Performs an atomic AND operation on the specified char values.
InterlockedCompareExchange	Performs an atomic compare-and-exchange operation on the specified values. The function compares two specified 32-bit values and exchanges with another 32-bit value based on the outcome of the comparison.
InterlockedCompareExchange128	Performs an atomic compare-and-exchange operation on the specified values. The function compares two specified 128-bit values and exchanges with another 128-bit value based on the outcome of the comparison.
InterlockedCompareExchange16	Performs an atomic compare-and-exchange operation on the specified values. The function compares two specified 16-bit values and exchanges with another 16-bit value based on the outcome of the comparison.
InterlockedCompareExchange64	Performs an atomic compare-and-exchange operation on the specified values. The function compares two specified 64-bit values and exchanges with another 64-bit value based on the outcome of the comparison.
InterlockedCompareExchangePointer	Performs an atomic compare-and-exchange operation on the specified values. The function compares two specified pointer values and exchanges with another pointer value based on the outcome of the comparison.
InterlockedDecrement	Decrements (decreases by one) the value of the specified 32-bit variable as an atomic operation.

InterlockedDecrement16
Decrements (decreases by one) the value of the specified 16-bit variable as an atomic operation.
InterlockedDecrement64
Decrements (decreases by one) the value of the specified 64-bit variable as an atomic operation.
InterlockedExchange
Sets a 32-bit variable to the specified value as an atomic operation.
InterlockedExchange16
Sets a 16-bit variable to the specified value as an atomic operation.
InterlockedExchange64
Sets a 64-bit variable to the specified value as an atomic operation.
InterlockedExchange8
Sets an 8-bit variable to the specified value as an atomic operation.
InterlockedExchangeAdd
Performs an atomic addition of two 32-bit values.
InterlockedExchangeAdd64
Performs an atomic addition of two 64-bit values.
InterlockedExchangePointer
Atomically exchanges a pair of addresses.
InterlockedIncrement
Increments (increases by one) the value of the specified 32-bit variable as an atomic operation.
InterlockedIncrement16
Increments (increases by one) the value of the specified 16-bit variable as an atomic operation.
InterlockedIncrement64
Increments (increases by one) the value of the specified 64-bit variable as an atomic operation.

InterlockedOr
Performs an atomic OR operation on the specified LONG values.
InterlockedOr16
Performs an atomic OR operation on the specified SHORT values.
InterlockedOr64
Performs an atomic OR operation on the specified LONGLONG values.
InterlockedOr8
Performs an atomic OR operation on the specified char values.
InterlockedXor
Performs an atomic XOR operation on the specified LONG values.
InterlockedXor16
Performs an atomic XOR operation on the specified SHORT values.
InterlockedXor64
Performs an atomic XOR operation on the specified LONGLONG values.
InterlockedXor8
Performs an atomic XOR operation on the specified char values.
IsReparseTagMicrosoft
Determines whether a reparse point tag indicates a Microsoft reparse point.
IsReparseTagNameSurrogate
Determines whether a tag's associated reparse point is a surrogate for another named entity (for example, a mounted folder).
LANGIDFROMLCID
Retrieves a language identifier from a locale identifier.
MAKELANGID
Creates a language identifier from a primary language identifier and a sublanguage identifier.

MAKELCID
Creates a locale identifier from a language identifier and a sort order identifier.
MAKESORTLCID
Constructs a locale identifier (LCID) from a language identifier, a sort order identifier, and the sort version.
MemoryBarrier
Creates a hardware memory barrier (fence) that prevents the CPU from re-ordering read and write operations. It may also prevent the compiler from re-ordering read and write operations.
Multiply128
Multiplies two 64-bit integers to produce a 128-bit integer.
MultiplyExtract128
Multiplies two 64-bit integers to produce a 128-bit integer, shifts the product to the right by the specified number of bits, and returns the low 64 bits of the result.
MultiplyHigh
Multiplies two 64-bit integers to produce a 128-bit integer and gets the high 64 bits.
NtCurrentTeb
The NtCurrentTeb routine returns a pointer to the Thread Environment Block (TEB) of the current thread.
PopulationCount64
Counts the number of one bits (population count) in a 64-bit unsigned integer.
PreFetchCacheLine
Indicates to the processor that a cache line will be needed in the near future.
PRIMARYLANGID
Extracts a primary language identifier from a language identifier.
RtlAddFunctionTable
Adds a dynamic function table to the dynamic function table list. (RtlAddFunctionTable)

RtlAddGrowableFunctionTable
Informs the system of a dynamic function table representing a region of memory containing code.
RtlCaptureContext
Retrieves a context record in the context of the caller.
RtlCaptureStackBackTrace
The RtlCaptureStackBackTrace routine captures a stack back trace by walking up the stack and recording the information for each frame.
RtlConvertDeviceFamilyInfoToString
Retrieves string representations of device family info.
RtlDeleteFunctionTable
Removes a dynamic function table from the dynamic function table list.
RtlDeleteGrowableFunctionTable
Informs the system that a previously reported dynamic function table is no longer in use.
RtlFirstEntrySList
Retrieves the first entry in a singly linked list. Access to the list is synchronized on a multiprocessor system.
RtlGrowFunctionTable
Reports that a dynamic function table has increased in size.
RtlInitializeSListHead
Initializes the head of a singly linked list. (RtlInitializeSListHead)
RtlInstallFunctionTableCallback
Adds a dynamic function table to the dynamic function table list. (RtlInstallFunctionTableCallback)
RtlInterlockedFlushSList
Removes all items from a singly linked list. Access to the list is synchronized on a multiprocessor system. (RtlInterlockedFlushSList)

[RtlInterlockedPopEntrySList](#)

Removes an item from the front of a singly linked list. Access to the list is synchronized on a multiprocessor system. (RtlInterlockedPopEntrySList)

[RtlInterlockedPushEntrySList](#)

Inserts an item at the front of a singly linked list. Access to the list is synchronized on a multiprocessor system. (RtlInterlockedPushEntrySList)

[RtlIsEcCode](#)

Returns a value indicating if the code pointed to by the supplied pointer is ARM emulation-compatible (ARM64EC).

[RtlLookupFunctionEntry](#)

Searches the active function tables for an entry that corresponds to the specified PC value.

[RtlPcToFileHeader](#)

Retrieves the base address of the image that contains the specified PC value.

[RtlQueryDepthSList](#)

Retrieves the number of entries in the specified singly linked list. (RtlQueryDepthSList)

[RtlRestoreContext](#)

Restores the context of the caller to the specified context record.

[RtlUnwind](#)

Initiates an unwind of procedure call frames. (RtlUnwind)

[RtlUnwind2](#)

Initiates an unwind of procedure call frames. (RtlUnwind2)

[RtlUnwindEx](#)

Initiates an unwind of procedure call frames.

[RtlVirtualUnwind](#)

Retrieves the invocation context of the function that precedes the specified function context.

[ShiftLeft128](#)

Shifts 128-bit left.

[ShiftRight128](#)

Shifts 128-bit right.

[SORTIDFROMLCID](#)

Retrieves a sort order identifier from a locale identifier.

[SORTVERSIONFROMLCID](#)

Retrieves the sort version from a locale identifier.

[SUBLANGID](#)

Extracts a sublanguage identifier from a language identifier.

[TEXT](#)

Identifies a string as Unicode when UNICODE is defined by a preprocessor directive during compilation. Otherwise, the macro identifies a string as an ANSI string.

[TpDestroyCallbackEnviron](#)

Deletes the specified callback environment. Call this function when the callback environment is no longer needed for creating new thread pool objects. (TpDestroyCallbackEnviron)

[TpInitializeCallbackEnviron](#)

Initializes a callback environment for the thread pool.

[TpSetCallbackActivationContext](#)

Assigns an activation context to the callback environment.

[TpSetCallbackCleanupGroup](#)

Associates the specified cleanup group with the specified callback environment.
(TpSetCallbackCleanupGroup)

[TpSetCallbackFinalizationCallback](#)

Indicates a function to call when the callback environment is finalized.

<p>TpSetCallbackLongFunction</p> <p>Indicates that callbacks associated with this callback environment may not return quickly. (TpSetCallbackLongFunction)</p>
<p>TpSetCallbackNoActivationContext</p> <p>Indicates that the callback environment has no activation context.</p>
<p>TpSetCallbackPersistent</p> <p>Specifies that the callback should run on a persistent thread. (TpSetCallbackPersistent)</p>
<p>TpSetCallbackPriority</p> <p>Specifies the priority of a callback function relative to other work items in the same thread pool. (TpSetCallbackPriority)</p>
<p>TpSetCallbackRaceWithDll</p> <p>Ensures that the specified DLL remains loaded as long as there are outstanding callbacks. (TpSetCallbackRaceWithDll)</p>
<p>TpSetCallbackThreadpool</p> <p>Assigns a thread pool to a callback environment.</p>
<p>UInt32x32To64</p> <p>Multiplies two unsigned 32-bit integers, returning an unsigned 64-bit integer result.</p>
<p>UnsignedMultiply128</p> <p>Multiplies two unsigned 64-bit integers to produce an unsigned 128-bit integer.</p>
<p>UnsignedMultiplyExtract128</p> <p>Multiplies two unsigned 64-bit integers to produce an unsigned 128-bit integer, shifts the product to the right by the specified number of bits, and returns the low 64 bits of the result.</p>
<p>UnsignedMultiplyHigh</p> <p>Multiplies two 64-bit integers to produce a 128-bit integer and gets the high unsigned 64 bits.</p>
<p>VER_SET_CONDITION</p> <p>Sets the bits of a 64-bit value to indicate the comparison operator to use for a specified operating system version attribute. This macro is used to build the dwlConditionMask parameter of the VerifyVersionInfo function.</p>

[VerSetConditionMask](#)

Sets the bits of a 64-bit value to indicate the comparison operator to use for a specified operating system version attribute. This function is used to build the dwlConditionMask parameter of the VerifyVersionInfo function.

[YieldProcessor](#)

Signals to the processor to give resources to threads that are waiting for them.

Callback functions

[PAPCFUNC](#)

An application-defined completion routine. Specify this address when calling the QueueUserAPC function.

[PFLS_CALLBACK_FUNCTION](#)

An application-defined function. If the FLS slot is in use, FlsCallback is called on fiber deletion, thread exit, and when an FLS index is freed.

[PSECURE_MEMORY_CACHE_CALLBACK](#)

An application-defined function previously registered with the AddSecureMemoryCacheCallback function that is called when a secured memory range is freed or its protections are changed.

[PVECTORED_EXCEPTION_HANDLER](#)

An application-defined function that serves as a vectored exception handler.

[RTL_UMS_SCHEDULER_ENTRY_POINT](#)

The application-defined user-mode scheduling (UMS) scheduler entry point function associated with a UMS completion list.

Structures

[ACCESS_ALLOWED_ACE](#)

Defines an access control entry (ACE) for the discretionary access control list (DACL) that controls access to an object. An access-allowed ACE allows access to an object for a specific trustee identified by a security identifier (SID).

[ACCESS_ALLOWED_CALLBACK_ACE](#)

The ACCESS_ALLOWED_CALLBACK_ACE structure defines an access control entry for the discretionary access control list that controls access to an object.

[ACCESS_ALLOWED_CALLBACK_OBJECT_ACE](#)

Defines an access control entry (ACE) that controls allowed access to an object, property set, or property.

[ACCESS_ALLOWED_OBJECT_ACE](#)

Defines an access control entry (ACE) that controls allowed access to an object, a property set, or property.

[ACCESS_DENIED_ACE](#)

Defines an access control entry (ACE) for the discretionary access control list (DACL) that controls access to an object. An access-denied ACE denies access to an object for a specific trustee identified by a security identifier (SID).

[ACCESS_DENIED_CALLBACK_ACE](#)

The ACCESS_DENIED_CALLBACK_ACE structure defines an access control entry for the discretionary access control list that controls access to an object.

[ACCESS_DENIED_CALLBACK_OBJECT_ACE](#)

The ACCESS_DENIED_CALLBACK_OBJECT_ACE structure defines an access control entry that controls denied access to an object, a property set, or property.

[ACCESS_DENIED_OBJECT_ACE](#)

Defines an access control entry (ACE) that controls denied access to an object, a property set, or property.

[ACE_HEADER](#)

Defines the type and size of an access control entry (ACE).

[ACL](#)

Header of an access control list (ACL).

[ACL_REVISION_INFORMATION](#)

Contains revision information about an ACL structure.

[ACL_SIZE_INFORMATION](#)

Contains information about the size of an ACL structure.

[ACTIVATION_CONTEXT_ASSEMBLY_DETAILED_INFORMATION](#)

The ACTIVATION_CONTEXT_ASSEMBLY_DETAILED_INFORMATION structure is used by the QueryActCtxW function.

[ACTIVATION_CONTEXT_COMPATIBILITY_INFORMATION](#)

The ACTIVATION_CONTEXT_COMPATIBILITY_INFORMATION structure is used by the QueryActCtxW function.

[ACTIVATION_CONTEXT_DETAILED_INFORMATION](#)

The ACTIVATION_CONTEXT_DETAILED_INFORMATION structure is used by the QueryActCtxW function.

[ACTIVATION_CONTEXT_QUERY_INDEX](#)

The ACTIVATION_CONTEXT_QUERY_INDEX structure is used by QueryActCtxW function.

[ACTIVATION_CONTEXT_RUN_LEVEL_INFORMATION](#)

The ACTIVATION_CONTEXT_RUN_LEVEL_INFORMATION structure is used by the QueryActCtxW function.

[ADMINISTRATOR_POWER_POLICY](#)

Represents the administrator override power policy settings.

[ARM64_NT_CONTEXT](#)

Contains processor-specific register data. The system uses CONTEXT structures to perform various internal operations.C

[ASSEMBLY_FILE_DETAILED_INFORMATION](#)

The ASSEMBLY_FILE_DETAILED_INFORMATION structure is used by the QueryActCtxW function.

[BATTERY_REPORTING_SCALE](#)

Contains the granularity of the battery capacity that is reported by IOCTL_BATTERY_QUERY_STATUS.

<p>CACHE_DESCRIPTOR</p> <p>Describes the cache attributes.</p>
<p>CACHE_RELATIONSHIP</p> <p>Describes cache attributes. This structure is used with the GetLogicalProcessorInformationEx function.</p>
<p>CLAIM_SECURITY_ATTRIBUTE_FQBN_VALUE</p> <p>Specifies the fully qualified binary name.</p>
<p>CLAIM_SECURITY_ATTRIBUTE_OCTET_STRING_VALUE</p> <p>Specifies the OCTET_STRING value type of the claim security attribute.</p>
<p>CLAIM_SECURITY_ATTRIBUTE_RELATIVE_V1</p> <p>Defines a resource attribute that is defined in continuous memory for persistence within a serialized security descriptor.</p>
<p>CLAIM_SECURITY_ATTRIBUTE_V1</p> <p>Defines a security attribute that can be associated with a token or authorization context.</p>
<p>CLAIM_SECURITY_ATTRIBUTES_INFORMATION</p> <p>Defines the security attributes for the claim.</p>
<p>COMPATIBILITY_CONTEXT_ELEMENT</p> <p>The COMPATIBILITY_CONTEXT_ELEMENT structure is used by the QueryActCtxW function as part of the ACTIVATION_CONTEXT_COMPATIBILITY_INFORMATION structure.</p>
<p>CONTEXT</p> <p>Contains processor-specific register data. The system uses CONTEXT structures to perform various internal operations. (CONTEXT)</p>
<p>ENCLAVE_CREATE_INFO_SGX</p> <p>Contains architecture-specific information to use to create an enclave when the enclave type is ENCLAVE_TYPE_SGX, which specifies an enclave for the Intel Software Guard Extensions (SGX) architecture extension.</p>

[ENCLAVE_CREATE_INFO_VBS](#)

Contains architecture-specific information to use to create an enclave when the enclave type is ENCLAVE_TYPE_VBS, which specifies a virtualization-based security (VBS) enclave.

[ENCLAVE_INIT_INFO_SGX](#)

Contains architecture-specific information to use to initialize an enclave when the enclave type is ENCLAVE_TYPE_SGX, which specifies an enclave for the Intel Software Guard Extensions (SGX) architecture extension.

[ENCLAVE_INIT_INFO_VBS](#)

Contains architecture-specific information to use to initialize an enclave when the enclave type is ENCLAVE_TYPE_VBS, which specifies a virtualization-based security (VBS) enclave.

[EVENTLOGRECORD](#)

Contains information about an event record returned by the ReadEventLog function.

[EXCEPTION_POINTERS](#)

Contains an exception record with a machine-independent description of an exception and a context record with a machine-dependent description of the processor context at the time of the exception.

[EXCEPTION_RECORD](#)

Describes an exception. (EXCEPTION_RECORD)

[EXCEPTION_RECORD64](#)

Describes an exception.E

[FILE_ID_128](#)

Defines a 128-bit file identifier.

[FILE_NOTIFY_EXTENDED_INFORMATION](#)

Describes the changes found by the ReadDirectoryChangesExW function.

[FILE_NOTIFY_INFORMATION](#)

Describes the changes found by the ReadDirectoryChangesW function.

<p>FILE_SEGMENT_ELEMENT</p> <p>The FILE_SEGMENT_ELEMENT structure represents a segment buffer structure for scatter/gather read/write actions.</p>
<p>FPO_DATA</p> <p>Represents the stack frame layout for a function on an x86 computer when frame pointer omission (FPO) optimization is used. The structure is used to locate the base of the call frame.</p>
<p>GENERIC_MAPPING</p> <p>Defines the mapping of generic access rights to specific and standard access rights for an object.</p>
<p>GROUP_AFFINITY</p> <p>Represents a processor group-specific affinity, such as the affinity of a thread.</p>
<p>GROUP_RELATIONSHIP</p> <p>Represents information about processor groups. This structure is used with the GetLogicalProcessorInformationEx function.</p>
<p>HARDWARE_COUNTER_DATA</p> <p>Contains the hardware counter value.</p>
<p>HEAP_OPTIMIZE_RESOURCES_INFORMATION</p> <p>Specifies flags for a HeapOptimizeResources operation initiated with HeapSetInformation.</p>
<p>IMAGE_COFF_SYMBOLS_HEADER</p> <p>Represents the COFF symbols header.</p>
<p>IMAGE_DATA_DIRECTORY</p> <p>Represents the data directory.</p>
<p>IMAGE_DEBUG_DIRECTORY</p> <p>Represents the debug directory format.</p>
<p>IMAGE_ENCLAVE_CONFIG32</p> <p>Defines the format of the enclave configuration for systems running 32-bit Windows. (32 bit)</p>

[IMAGE_ENCLAVE_CONFIG64](#)

Defines the format of the enclave configuration for systems running 32-bit Windows. (64 bit)

[IMAGE_ENCLAVE_IMPORT](#)

Defines a entry in the array of images that an enclave can import.

[IMAGE_FILE_HEADER](#)

Represents the COFF header format.

[IMAGE_FUNCTION_ENTRY](#)

Represents an entry in the function table. (IMAGE_FUNCTION_ENTRY)

[IMAGE_FUNCTION_ENTRY64](#)

Represents an entry in the function table.

[IMAGE_LOAD_CONFIG_DIRECTORY32](#)

Contains the load configuration data of an image. (32 bit)

[IMAGE_LOAD_CONFIG_DIRECTORY64](#)

Contains the load configuration data of an image. (64 bit)

[IMAGE_NT_HEADERS32](#)

Represents the PE header format. (32 bit)

[IMAGE_NT_HEADERS64](#)

Represents the PE header format. (64 bit)

[IMAGE_OPTIONAL_HEADER32](#)

Represents the optional header format. (32 bit)

[IMAGE_OPTIONAL_HEADER64](#)

Represents the optional header format. (64 bit)

[IMAGE_SECTION_HEADER](#)

Represents the image section header format.

IO_COUNTERS
Contains I/O accounting information for a process or a job object.
JOBOBJECT_ASSOCIATE_COMPLETION_PORT
Contains information used to associate a completion port with a job.
JOBOBJECT_BASIC_ACCOUNTING_INFORMATION
Contains basic accounting information for a job object.
JOBOBJECT_BASIC_AND_IO_ACCOUNTING_INFORMATION
Contains basic accounting and I/O accounting information for a job object.
JOBOBJECT_BASIC_LIMIT_INFORMATION
Contains basic limit information for a job object.
JOBOBJECT_BASIC_PROCESS_ID_LIST
Contains the process identifier list for a job object.
JOBOBJECT_BASIC_UI_RESTRICTIONS
Contains basic user-interface restrictions for a job object.
JOBOBJECT_CPU_RATE_CONTROL_INFORMATION
Contains CPU rate control information for a job object. This structure is used by the SetInformationJobObject and QueryInformationJobObject functions with the JobObjectCpuRateControlInformation information class.
JOBOBJECT_END_OF_JOB_TIME_INFORMATION
Specifies the action the system will perform when an end-of-job time limit is exceeded.
JOBOBJECT_EXTENDED_LIMIT_INFORMATION
Contains basic and extended limit information for a job object.
JOBOBJECT_LIMIT_VIOLATION_INFORMATION
Contains information about resource notification limits that have been exceeded for a job object. This structure is used with the QueryInformationJobObject function with the JobObjectLimitViolationInformation information class.

[JOB_OBJECT_LIMIT_VIOLATION_INFORMATION_2](#)

Contains extended information about resource notification limits that have been exceeded for a job object. This structure is used with the `QueryInformationJobObject` function with the `JobObjectLimitViolationInformation2` information class.

[JOB_OBJECT_NET_RATE_CONTROL_INFORMATION](#)

Contains information used to control the network traffic for a job. This structure is used by the `SetInformationJobObject` and `QueryInformationJobObject` functions with the `JobObjectNetRateControlInformation` information class.

[JOB_OBJECT_NOTIFICATION_LIMIT_INFORMATION](#)

Contains information about notification limits for a job object. This structure is used by the `SetInformationJobObject` and `QueryInformationJobObject` functions with the `JobObjectNotificationLimitInformation` information class.

[JOB_OBJECT_NOTIFICATION_LIMIT_INFORMATION_2](#)

Contains extended information about notification limits for a job object. This structure is used by the `SetInformationJobObject` and `QueryInformationJobObject` functions with the `JobObjectNotificationLimitInformation2` information class.

[JOB_OBJECT_SECURITY_LIMIT_INFORMATION](#)

Contains the security limitations for a job object.

[LARGE_INTEGER](#)

The `LARGE_INTEGER` structure represents a 64-bit signed integer value. (`LARGE_INTEGER` union (`winnt.h`))

[LDT_ENTRY](#)

Describes an entry in the descriptor table. This structure is valid only on x86-based systems.

[LUID](#)

Describes a local identifier for an adapter. (`LUID`)

[LUID_AND_ATTRIBUTES](#)

Represents a locally unique identifier (`LUID`) and its attributes.

[MEM_ADDRESS_REQUIREMENTS](#)

Specifies a lowest and highest base address and alignment as part of an extended parameter to a function that manages virtual memory.

<p>MEM_EXTENDED_PARAMETER</p> <p>Represents an extended parameter for a function that manages virtual memory.</p>
<p>MEMORY_BASIC_INFORMATION</p> <p>Contains information about a range of pages in the virtual address space of a process.</p>
<p>MESSAGE_RESOURCE_BLOCK</p> <p>Contains information about message strings with identifiers in the range indicated by the LowId and HighId members.</p>
<p>MESSAGE_RESOURCE_DATA</p> <p>Contains information about formatted text for display as an error message or in a message box in a message table resource.</p>
<p>MESSAGE_RESOURCE_ENTRY</p> <p>Contains the error message or message box display text for a message table resource.</p>
<p>NUMA_NODE_RELATIONSHIP</p> <p>Represents information about a NUMA node in a processor group. This structure is used with the GetLogicalProcessorInformationEx function.</p>
<p>OBJECT_TYPE_LIST</p> <p>Identifies an object type element in a hierarchy of object types.</p>
<p>OSVERSIONINFOA</p> <p>Contains operating system version information. (ANSI)</p>
<p>OSVERSIONINFOEXA</p> <p>Contains operating system version information. The information includes major and minor version numbers, a build number, a platform identifier, and information about product suites and the latest Service Pack installed on the system. (ANSI)</p>
<p>OSVERSIONINFOEXW</p> <p>Contains operating system version information. The information includes major and minor version numbers, a build number, a platform identifier, and information about product suites and the latest Service Pack installed on the system. (Unicode)</p>

[OSVERSIONINFOW](#)

Contains operating system version information. (Unicode)

[PERFORMANCE_DATA](#)

Contains the thread profiling and hardware counter data that you requested.

[POWER_ACTION_POLICY](#)

Contains information used to set the system power state.

[PRIVILEGE_SET](#)

Specifies a set of privileges.

[PROCESS_DYNAMIC_EH_CONTINUATION_TARGET](#)

Contains dynamic exception handling continuation targets.

[PROCESS_DYNAMIC_ENFORCED_ADDRESS_RANGE](#)

Contains dynamic enforced address ranges used by various features related to user-mode Hardware-enforced Stack Protection (HSP).

[PROCESS_MITIGATION_ASLR_POLICY](#)

Contains process mitigation policy settings for Address Space Randomization Layout (ASLR).

[PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY](#)

Contains process mitigation policy settings for the loading of images depending on the signatures for the image.

[PROCESS_MITIGATION_CONTROL_FLOW_GUARD_POLICY](#)

Contains process mitigation policy settings for Control Flow Guard (CFG).

[PROCESS_MITIGATION_DEP_POLICY](#)

Contains process mitigation policy settings for data execution prevention (DEP).

[PROCESS_MITIGATION_DYNAMIC_CODE_POLICY](#)

Contains process mitigation policy settings for restricting dynamic code generation and modification.

PROCESS_MITIGATION_EXTENSION_POINT_DISABLE_POLICY
Contains process mitigation policy settings for legacy extension point DLLs.
PROCESS_MITIGATION_FONT_DISABLE_POLICY
Contains process mitigation policy settings for the loading of non-system fonts.
PROCESS_MITIGATION_IMAGE_LOAD_POLICY
Contains process mitigation policy settings for the loading of images from a remote device.
PROCESS_MITIGATION_REDIRECTION_TRUST_POLICY
Contains process mitigation policy settings for the ???.
PROCESS_MITIGATION_SIDE_CHANNEL_ISOLATION_POLICY
This data structure provides the status of process policies that are related to the mitigation of side channels. This can include side channel attacks involving speculative execution and page combining.
PROCESS_MITIGATION_STRICT_HANDLE_CHECK_POLICY
Used to impose new behavior on handle references that are not valid.
PROCESS_MITIGATION_SYSTEM_CALL_DISABLE_POLICY
Used to impose restrictions on what system calls can be invoked by a process.
PROCESS_MITIGATION_USER_SHADOW_STACK_POLICY
Contains process mitigation policy settings for user-mode Hardware-enforced Stack Protection (HSP).
PROCESSOR_GROUP_INFO
Represents the number and affinity of processors in a processor group.
PROCESSOR_NUMBER
Represents a logical processor in a processor group.
PROCESSOR_POWER_POLICY
Contains information about processor performance control and C-states.

PROCESSOR_POWER_POLICY_INFO
Contains information about processor C-state policy settings.
PROCESSOR_RELATIONSHIP
Represents information about affinity within a processor group. This structure is used with the GetLogicalProcessorInformationEx function.
QUOTA_LIMITS
Describes the amount of system resources available to a user.
REPARSE_GUID_DATA_BUFFER
Contains information about a reparse point.
RUNTIME_FUNCTION
Represents an entry in the function table on 64-bit Windows.
SECURITY_CAPABILITIES
Defines the security capabilities of the app container.
SECURITY_DESCRIPTOR
Contains the security information associated with an object.
SECURITY_QUALITY_OF_SERVICE
Contains information used to support client impersonation.
SID
Used to uniquely identify users or groups.
SID_AND_ATTRIBUTES
Represents a security identifier (SID) and its attributes.
SID_AND_ATTRIBUTES_HASH
Specifies a hash values for the specified array of security identifiers (SIDs).
SID_IDENTIFIER_AUTHORITY
Represents the top-level authority of a security identifier (SID).

[SINGLE_LIST_ENTRY](#)

Represents an item in a singly linked list.

[SLIST_ENTRY](#)

Represents an item in a singly linked list. (SLIST_ENTRY)

[SYSTEM_ALARM_ACE](#)

The SYSTEM_ALARM_ACE structure is reserved for future use.

[SYSTEM_ALARM_CALLBACK_ACE](#)

The SYSTEM_ALARM_CALLBACK_ACE structure is reserved for future use.

[SYSTEM_ALARM_CALLBACK_OBJECT_ACE](#)

The SYSTEM_ALARM_CALLBACK_OBJECT_ACE structure is reserved for future use.

[SYSTEM_ALARM_OBJECT_ACE](#)

The SYSTEM_ALARM_OBJECT_ACE structure is reserved for future use.

[SYSTEM_AUDIT_ACE](#)

Defines an access control entry (ACE) for the system access control list (SACL) that specifies what types of access cause system-level notifications.

[SYSTEM_AUDIT_CALLBACK_ACE](#)

The SYSTEM_AUDIT_CALLBACK_ACE structure defines an access control entry for the system access control list that specifies what types of access cause system-level notifications.

[SYSTEM_AUDIT_CALLBACK_OBJECT_ACE](#)

The SYSTEM_AUDIT_CALLBACK_OBJECT_ACE structure defines an access control entry for a system access control list.

[SYSTEM_AUDIT_OBJECT_ACE](#)

Defines an access control entry (ACE) for a system access control list (SACL).

[SYSTEM_BATTERY_STATE](#)

Contains information about the current state of the system battery.

[SYSTEM_CPU_SET_INFORMATION](#)

This structure is returned by GetSystemCpuSetInformation. It is used to enumerate the CPU Sets on the system and determine their current state.

[SYSTEM_LOGICAL_PROCESSOR_INFORMATION](#)

Describes the relationship between the specified processor set. This structure is used with the GetLogicalProcessorInformation function.

[SYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX](#)

Contains information about the relationships of logical processors and related hardware. The GetLogicalProcessorInformationEx function uses this structure.

[SYSTEM_MANDATORY_LABEL_ACE](#)

Defines an access control entry (ACE) for the system access control list (SACL) that specifies the mandatory access level and policy for a securable object.

[SYSTEM_POWER_CAPABILITIES](#)

Contains information about the power capabilities of the system.

[SYSTEM_POWER_LEVEL](#)

Contains information about system battery drain policy settings.

[SYSTEM_POWER_POLICY](#)

Contains information about the current system power policy.

[SYSTEM_RESOURCE_ATTRIBUTE_ACE](#)

Defines an access control entry (ACE) for the system access control list (SACL) that specifies the system resource attributes for a securable object.

[SYSTEM_SCOPED_POLICY_ID_ACE](#)

Defines an access control entry (ACE) for the system access control list (SACL) that specifies the scoped policy identifier for a securable object.

[TAPE_ERASE](#)

Describes the partition to be erased.

[TAPE_GET_DRIVE_PARAMETERS](#)

Describes the tape drive. It is used by the GetTapeParameters function.

[TAPE_GET_MEDIA_PARAMETERS](#)

Describes the tape in the tape drive. It is used by the GetTapeParametersfunction.

[TAPE_GET_POSITION](#)

Describes the position of the tape.

[TAPE_PREPARE](#)

Describes how to prepare the tape.

[TAPE_SET_DRIVE_PARAMETERS](#)

Describes the tape drive. It is used by the SetTapeParametersfunction.

[TAPE_SET_MEDIA_PARAMETERS](#)

Describes the tape in the tape drive. It is used by the SetTapeParametersfunction.

[TAPE_SET_POSITION](#)

Describes how and where to position the tape.

[TAPE_WRITE_MARKS](#)

Describes the type and number of tapemarks to write.

[TOKEN_ACCESS_INFORMATION](#)

Specifies all the information in a token that is necessary to perform an access check.

[TOKEN_APPCONTAINER_INFORMATION](#)

Specifies all the information in a token that is necessary for an app container.

[TOKEN_AUDIT_POLICY](#)

Specifies the per user audit policy for a token.

[TOKEN_CONTROL](#)

Contains information that identifies an access token.

[TOKEN_DEFAULT_DACL](#)

Specifies a discretionary access control list (DACL).

TOKEN_DEVICE_CLAIMS

Defines the device claims for the token.

TOKEN_ELEVATION

Indicates whether a token has elevated privileges.

TOKEN_GROUPS

Contains information about the group security identifiers (SIDs) in an access token.

TOKEN_GROUPS_AND_PRIVILEGES

Contains information about the group security identifiers (SIDs) and privileges in an access token.

TOKEN_LINKED_TOKEN

Contains a handle to a token. This token is linked to the token being queried by the GetTokenInformation function or set by the SetTokenInformation function.

TOKEN_MANDATORY_LABEL

Specifies the mandatory integrity level for a token.

TOKEN_MANDATORY_POLICY

Specifies the mandatory integrity policy for a token.

TOKEN_ORIGIN

Contains information about the origin of the logon session.

TOKEN_OWNER

Contains the default owner security identifier (SID) that will be applied to newly created objects.

TOKEN_PRIMARY_GROUP

Specifies a group security identifier (SID) for an access token.

TOKEN_PRIVILEGES

Contains information about a set of privileges for an access token.

TOKEN_SOURCE

Identifies the source of an access token.

[TOKEN_STATISTICS](#)

Contains information about an access token.

[TOKEN_USER](#)

Identifies the user associated with an access token.

[TOKEN_USER_CLAIMS](#)

Defines the user claims for the token.

[ULARGE_INTEGER](#)

The ULARGE_INTEGER structure represents a 64-bit unsigned integer value. (ULARGE_INTEGER union (winnt.h))

[UMS_CREATE_THREAD_ATTRIBUTES](#)

Specifies attributes for a user-mode scheduling (UMS) worker thread.

[WOW64_CONTEXT](#)

Represents a context frame on WOW64.

[WOW64_FLOATING_SAVE_AREA](#)

Represents the 80387 save area on WOW64.

[WOW64_LDT_ENTRY](#)

Describes an entry in the descriptor table for a 32-bit thread on a 64-bit system. This structure is valid only on 64-bit systems.

Enumerations

[ACL_INFORMATION_CLASS](#)

Contains values that specify the type of information being assigned to or retrieved from an access control list (ACL).

[ACTCTX_COMPATIBILITY_ELEMENT_TYPE](#)

The ACTCTX_COMPATIBILITY_ELEMENT_TYPE enumeration describes the compatibility element in the application manifest.

[ACTCTX_REQUESTED_RUN_LEVEL](#)

The ACTCTX_REQUESTED_RUN_LEVEL enumeration describes the requested run level of the activation context.

[AUDIT_EVENT_TYPE](#)

Defines values that indicate the type of object being audited. The AccessCheckByTypeAndAuditAlarm and AccessCheckByTypeResultListAndAuditAlarm functions use these values.

[COMPARTMENT_ID](#)

The COMPARTMENT_ID enumeration indicates the network routing compartment identifier.

[FIRMWARE_TYPE](#)

Specifies a firmware type.

[HARDWARE_COUNTER_TYPE](#)

Defines the types of hardware counters being profiled.

[HEAP_INFORMATION_CLASS](#)

Specifies the class of heap information to be set or retrieved.

[JOB_OBJECT_NET_RATE_CONTROL_FLAGS](#)

Specifies types of scheduling policies for network rate control.

[LOGICAL_PROCESSOR_RELATIONSHIP](#)

Represents the relationship between the processor set identified in the corresponding SYSTEM_LOGICAL_PROCESSOR_INFORMATION or SYSTEM_LOGICAL_PROCESSOR_INFORMATION_EX structure.

[MANDATORY_LEVEL](#)

Lists the possible security levels.

[MEM_EXTENDED_PARAMETER_TYPE](#)

Defines values for extended parameters used for file mapping into an address space.

[POWER_ACTION](#)

Defines values that are used to specify system power action types.

POWER_PLATFORM_ROLE
Indicates the OEM's preferred power management profile.
PROCESS_MITIGATION_POLICY
Represents the different process mitigation policies.
PROCESSOR_CACHE_TYPE
Represents the type of processor cache identified in the corresponding CACHE_DESCRIPTOR structure.
SECURITY_IMPERSONATION_LEVEL
Contains values that specify security impersonation levels. Security impersonation levels govern the degree to which a server process can act on behalf of a client process.
SID_NAME_USE
Contains values that specify the type of a security identifier (SID).
SYSTEM_POWER_CONDITION
Used by the GUID_ACDC_POWER_SOURCE power event to indicate the current power source.
SYSTEM_POWER_STATE
Defines values that are used to specify system power states.
TOKEN_ELEVATION_TYPE
Indicates the elevation type of token being queried by the GetTokenInformation function or set by the SetTokenInformation function.
TOKEN_INFORMATION_CLASS
Contains values that specify the type of information being assigned to or retrieved from an access token.
TOKEN_TYPE
Contains values that differentiate between a primary token and an impersonation token.
TRANSACTION_OUTCOME
Defines the outcomes (results) that KTM can assign to a transaction.

[WELL_KNOWN_SID_TYPE](#)

A list of commonly used security identifiers (SIDs). Programs can pass these values to the CreateWellKnownSid function to create a SID from this list.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

LUID structure (winnt.h)

Article 07/27/2022

Describes a local identifier for an adapter.

Syntax

C++

```
typedef struct _LUID {
    DWORD LowPart;
    LONG HighPart;
} LUID, *PLUID;
```

Members

LowPart

Specifies a DWORD that contains the unsigned lower numbers of the id.

HighPart

Specifies a LONG that contains the signed high numbers of the id.

Remarks

This structure is used by the [ID3D12Device::GetAdapterLuid](#) and [GetSharedResourceAdapterLuid](#) methods.

Requirements

Header

winnt.h

See also

[DXGI Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)