

Direct3D 12 Graphics

Article 02/14/2023

Overview of the Direct3D 12 Graphics technology.

To develop Direct3D 12 Graphics, you need these headers:

- [d3d11on12.h](#)
- [d3d12.h](#)
- [d3d12sdklayers.h](#)
- [d3d12shader.h](#)
- [windows.graphics.holographic.interop.h](#)

For programming guidance for this technology, see:

- [Direct3D 12 Graphics](#)

Enumerations

[+] [Expand table](#)

D3D_ROOT_SIGNATURE_VERSION
Specifies the version of root signature layout.
D3D_SHADER_MODEL
Specifies a shader model.
D3D12_AUTO_BREADCRUMB_OP
Defines constants that specify render/compute GPU operations. (D3D12_AUTO_BREADCRUMB_OP)
D3D12_AXIS_SHADING_RATE
Defines constants that specify the shading rate (for variable-rate shading, or VRS) along a horizontal or vertical axis.
D3D12_BACKGROUND_PROCESSING_MODE
Defines constants that specify a level of dynamic optimization to apply to GPU work that's subsequently submitted.

[D3D12_BARRIER_ACCESS](#)

[D3D12_BARRIER_LAYOUT](#)

[D3D12_BARRIER_SYNC](#)

[D3D12_BARRIER_TYPE](#)

[D3D12_BLEND](#)

Specifies blend factors, which modulate values for the pixel shader and render target.

[D3D12_BLEND_OP](#)

Specifies RGB or alpha blending operations.

[D3D12_BUFFER_SRV_FLAGS](#)

Identifies how to view a buffer resource. (D3D12_BUFFER_SRV_FLAGS)

[D3D12_BUFFER_UAV_FLAGS](#)

Identifies unordered-access view options for a buffer resource. (D3D12_BUFFER_UAV_FLAGS)

[D3D12_CLEAR_FLAGS](#)

Specifies what to clear from the depth stencil view.

[D3D12_COLOR_WRITE_ENABLE](#)

Identifies which components of each pixel of a render target are writable during blending.

[D3D12_COMMAND_LIST_FLAGS](#)

The D3D12_COMMAND_LIST_FLAGS enumeration specifies flags to be used when creating a command list.

[D3D12_COMMAND_LIST_SUPPORT_FLAGS](#)

Used to determine which kinds of command lists are capable of supporting various operations.

[D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of a command list.

D3D12_COMMAND_POOL_FLAGS
D3D12_COMMAND_QUEUE_FLAGS
Specifies flags to be used when creating a command queue.
D3D12_COMMAND_QUEUE_PRIORITY
Defines priority levels for a command queue.
D3D12_COMMAND_RECORDER_FLAGS
D3D12_COMPARISON_FUNC
Specifies comparison options.
D3D12_CONSERVATIVE_RASTERIZATION_MODE
Identifies whether conservative rasterization is on or off. (D3D12_CONSERVATIVE_RASTERIZATION_MODE)
D3D12_CONSERVATIVE_RASTERIZATION_TIER
Identifies the tier level of conservative rasterization.
D3D12_CPU_PAGE_PROPERTY
Specifies the CPU-page properties for the heap.
D3D12_CROSS_NODE_SHARING_TIER
Specifies the level of sharing across nodes of an adapter, such as Tier 1 Emulated, Tier 1, or Tier 2.
D3D12_CULL_MODE
Specifies triangles facing a particular direction are not drawn.
D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE
Indicates the debug parameter type used by ID3D12DebugCommandList1::SetDebugParameter and ID3D12DebugCommandList1::GetDebugParameter.
D3D12_DEBUG_DEVICE_PARAMETER_TYPE
Specifies the data type of the memory pointed to by the pData parameter of ID3D12DebugDevice1::SetDebugParameter and ID3D12DebugDevice1::GetDebugParameter.

D3D12_DEBUG_FEATURE	Flags for optional D3D12 Debug Layer features.
D3D12_DEPTH_WRITE_MASK	Identifies the portion of a depth-stencil buffer for writing depth data.
D3D12_DESCRIPTOR_HEAP_FLAGS	Specifies options for a heap.
D3D12_DESCRIPTOR_HEAP_TYPE	Specifies a type of descriptor heap.
D3D12_DESCRIPTOR_RANGE_FLAGS	Specifies the volatility of both descriptors and the data they reference in a Root Signature 1.1 description, which can enable some driver optimizations.
D3D12_DESCRIPTOR_RANGE_TYPE	Specifies a range so that, for example, if part of a descriptor table has 100 shader-resource views (SRVs) that range can be declared in one entry rather than 100.
D3D12_DRED_ALLOCATION_TYPE	Congruent with, and numerically equivalent to, 3D12DDI_HANDLETYPE enumeration values.
D3D12_DRED_DEVICE_STATE	
D3D12_DRED_ENABLEMENT	Defines constants that specify render/compute GPU operations. (D3D12_DRED_ENABLEMENT)
D3D12_DRED_FLAGS	Defines constants used in the D3D12_DEVICE_REMOVED_EXTENDED_DATA structure to specify control flags for the Direct3D runtime.
D3D12_DRED_PAGEFAULT_FLAGS	
D3D12_DRED_VERSION	Defines constants that specify a version of Device Removed Extended Data (DRED), as used by the D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure.

D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS	Specifies the result of a call to ID3D12Device5::CheckDriverMatchingIdentifier which queries whether serialized data is compatible with the current device and driver version.
D3D12_DSV_DIMENSION	Specifies how to access a resource used in a depth-stencil view. (D3D12_DSV_DIMENSION)
D3D12_DSV_FLAGS	Specifies depth-stencil view options.
D3D12_ELEMENTS_LAYOUT	Describes how the locations of elements are identified.
D3D12_EXPORT_FLAGS	The flags to apply when exporting symbols from a state subobject.
D3D12_FEATURE	Defines constants that specify a Direct3D 12 feature or feature set to query about.
D3D12_FENCE_FLAGS	Specifies fence options. (D3D12_FENCE_FLAGS)
D3D12_FILL_MODE	Specifies the fill mode to use when rendering triangles.
D3D12_FILTER	Specifies filtering options during texture sampling.
D3D12_FILTER_REDUCTION_TYPE	Specifies the type of filter reduction.
D3D12_FILTER_TYPE	Specifies the type of magnification or minification sampler filters.
D3D12_FORMAT_SUPPORT1	Specifies resources that are supported for a provided format.

D3D12_FORMAT_SUPPORT2	Specifies which unordered resource options are supported for a provided format.
D3D12_GPU_BASED_VALIDATION_FLAGS	Describes the level of GPU-based validation to perform at runtime.
D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS	Specifies how GPU-Based Validation handles patched pipeline states during ID3D12Device::CreateGraphicsPipelineState and ID3D12Device::CreateComputePipelineState.
D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE	Specifies the type of shader patching used by GPU-Based Validation at either the device or command list level.
D3D12_GRAPHICS_STATES	Defines flags that specify states related to a graphics command list. Values can be bitwise OR'd together.
D3D12_HEAP_FLAGS	Specifies heap options, such as whether the heap can contain textures, and whether resources are shared across adapters.
D3D12_HEAP_SERIALIZATION_TIER	Defines constants that specify heap serialization support.
D3D12_HEAP_TYPE	Specifies the type of heap. When resident, heaps reside in a particular physical memory pool with certain CPU cache properties.
D3D12_HIT_GROUP_TYPE	Specifies the type of a raytracing hit group state subobject. Use a value from this enumeration with the D3D12_HIT_GROUP_DESC structure.
D3D12_HIT_KIND	
D3D12_INDEX_BUFFER_STRIP_CUT_VALUE	When using triangle strip primitive topology, vertex positions are interpreted as vertices of a continuous triangle "strip".

D3D12_INDIRECT_ARGUMENT_TYPE	Specifies the type of the indirect parameter.
D3D12_INPUT_CLASSIFICATION	Identifies the type of data contained in an input slot.
D3D12_LIFETIME_STATE	Defines constants that specify the lifetime state of a lifetime-tracked object.
D3D12_LOGIC_OP	Specifies logical operations to configure for a render target. (D3D12_LOGIC_OP)
D3D12_MEASUREMENTS_ACTION	Defines constants that specify what should be done with the results of earlier workload instrumentation.
D3D12_MEMORY_POOL	Specifies the memory pool for the heap.
D3D12_MESH_SHADER_TIER	Defines constants that specify mesh and amplification shader support.
D3D12_MESSAGE_CATEGORY	Specifies categories of debug messages.
D3D12_MESSAGE_ID	Specifies debug message IDs for setting up an info-queue filter (see D3D12_INFO_QUEUE_FILTER); use these messages to allow or deny message categories to pass through the storage and retrieval filters.
D3D12_MESSAGE_SEVERITY	Debug message severity levels for an information queue. (D3D12_MESSAGE_SEVERITY)
D3D12_META_COMMAND_PARAMETER_FLAGS	Defines constants that specify the flags for a parameter to a meta command. Values can be bitwise OR'd together.

[D3D12_META_COMMAND_PARAMETER_STAGE](#)

Defines constants that specify the stage of a parameter to a meta command.

[D3D12_META_COMMAND_PARAMETER_TYPE](#)

Defines constants that specify the data type of a parameter to a meta command.

[D3D12_MULTIPLE_FENCE_WAIT_FLAGS](#)

Specifies multiple wait flags for multiple fences.

[D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS](#)

Specifies options for determining quality levels.

[D3D12_PIPELINE_STATE_FLAGS](#)

Flags to control pipeline state.

[D3D12_PIPELINE_STATE_SUBOBJECT_TYPE](#)

Specifies the type of a sub-object in a pipeline state stream description.

[D3D12_PREDICATION_OP](#)

Specifies the predication operation to apply.

[D3D12_PRIMITIVE_TOPOLOGY_TYPE](#)

Specifies how the pipeline interprets geometry or hull shader input primitives.

[D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER](#)

Specifies the level of support for programmable sample positions that's offered by the adapter.

[D3D12_PROTECTED_RESOURCE_SESSION_FLAGS](#)

Defines constants that specify protected resource session flags.

[D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAGS](#)

Defines constants that specify protected resource session support.

[D3D12_PROTECTED_SESSION_STATUS](#)

Defines constants that specify protected session status.

[D3D12_QUERY_HEAP_TYPE](#)

	Specifies the type of query heap to create.
D3D12_QUERY_TYPE	Specifies the type of query.
D3D12_RAY_FLAGS	Flags passed to the TraceRay function to override transparency, culling, and early-out behavior.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS	Specifies flags for the build of a raytracing acceleration structure. Use a value from this enumeration with the D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS structure that provides input to the acceleration structure build operation.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE	Specifies the type of copy operation performed when calling CopyRaytracingAccelerationStructure.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE	Specifies the type of acceleration structure post-build info that can be retrieved with calls to EmitRaytracingAccelerationStructurePostbuildInfo and BuildRaytracingAccelerationStructure.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE	Specifies the type of a raytracing acceleration structure.
D3D12_RAYTRACING_GEOMETRY_FLAGS	Specifies flags for raytracing geometry in a D3D12_RAYTRACING_GEOMETRY_DESC structure.
D3D12_RAYTRACING_GEOMETRY_TYPE	Specifies the type of geometry used for raytracing. Use a value from this enumeration to specify the geometry type in a D3D12_RAYTRACING_GEOMETRY_DESC.
D3D12_RAYTRACING_INSTANCE_FLAGS	Flags for a raytracing acceleration structure instance. These flags can be used to override D3D12_RAYTRACING_GEOMETRY_FLAGS for individual instances.
D3D12_RAYTRACING_PIPELINE_FLAGS	Defines constants that specify configuration flags for a raytracing pipeline.
D3D12_RAYTRACING_TIER	

	Specifies the level of ray tracing support on the graphics device.
D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE	Specifies the type of access that an application is given to the specified resource(s) at the transition into a render pass.
D3D12_RENDER_PASS_ENDING_ACCESS_TYPE	Specifies the type of access that an application is given to the specified resource(s) at the transition out of a render pass.
D3D12_RENDER_PASS_FLAGS	Specifies the nature of the render pass; for example, whether it is a suspending or a resuming render pass.
D3D12_RENDER_PASS_TIER	Specifies the level of support for render passes on a graphics device.
D3D12_RESIDENCY_FLAGS	Used with the EnqueueMakeResident function to choose how residency operations proceed when the memory budget is exceeded.
D3D12_RESIDENCY_PRIORITY	Specifies broad residency priority buckets useful for quickly establishing an application priority scheme.
D3D12_RESOLVE_MODE	Specifies a resolve operation.
D3D12_RESOURCE_BARRIER_FLAGS	Flags for setting split resource barriers.
D3D12_RESOURCE_BARRIER_TYPE	Specifies a type of resource barrier (transition in resource use) description.
D3D12_RESOURCE_BINDING_TIER	Identifies the tier of resource binding being used.
D3D12_RESOURCE_DIMENSION	

Identifies the type of resource being used. (D3D12_RESOURCE_DIMENSION)
D3D12_RESOURCE_FLAGS
Specifies options for working with resources.
D3D12_RESOURCE_HEAP_TIER
Specifies which resource heap tier the hardware and driver support.
D3D12_RESOURCE_STATES
Defines constants that specify the state of a resource regarding how the resource is being used.
D3D12_RLDO_FLAGS
Specifies options for the amount of information to report about a live device object's lifetime.
D3D12_ROOT_DESCRIPTOR_FLAGS
Specifies the volatility of the data referenced by descriptors in a Root Signature 1.1 description, which can enable some driver optimizations.
D3D12_ROOT_PARAMETER_TYPE
Specifies the type of root signature slot.
D3D12_ROOT_SIGNATURE_FLAGS
Specifies options for root signature layout.
D3D12_RTV_DIMENSION
Identifies the type of resource to view as a render target.
D3D12_SAMPLER_FEEDBACK_TIER
Defines constants that specify sampler feedback support.
D3D12_SERIALIZED_DATA_TYPE
Specifies the type of serialized data. Use a value from this enumeration when calling ID3D12Device5::CheckDriverMatchingIdentifier.
D3D12_SHADER_CACHE_CONTROL_FLAGS
Defines constants that specify shader cache control options.
D3D12_SHADER_CACHE_FLAGS

Defines constants that specify shader cache flags.
D3D12_SHADER_CACHE_KIND_FLAGS
Defines constants that specify a kind of shader cache.
D3D12_SHADER_CACHE_MODE
Defines constants that specify a shader cache's mode.
D3D12_SHADER_CACHE_SUPPORT_FLAGS
Describes the level of support for shader caching in the current graphics driver. (D3D12_SHADER_CACHE_SUPPORT_FLAGS)
D3D12_SHADER_COMPONENT_MAPPING
Specifies how memory gets routed by a shader resource view (SRV).
D3D12_SHADER_MIN_PRECISION_SUPPORT
Describes minimum precision support options for shaders in the current graphics driver.
D3D12_SHADER_VERSION_TYPE
Enumerates the types of shaders that Direct3D recognizes. Used to encode the Version member of the D3D12_SHADER_DESC structure.
D3D12_SHADER_VISIBILITY
Specifies the shaders that can access the contents of a given root signature slot.
D3D12_SHADING_RATE
Defines constants that specify the shading rate (for variable-rate shading, or VRS).
D3D12_SHADING_RATE_COMBINER
Defines constants that specify a shading rate combiner (for variable-rate shading, or VRS).
D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER
Defines constants that specify a cross-API sharing support tier.
D3D12_SRV_DIMENSION
Identifies the type of resource that will be viewed as a shader resource.
D3D12_STATE_OBJECT_FLAGS

	Specifies constraints for state objects. Use values from this enumeration in the D3D12_STATE_OBJECT_CONFIG structure.
D3D12_STATE_OBJECT_TYPE	Specifies the type of a state object. Use with D3D12_STATE_OBJECT_DESC.
D3D12_STATE_SUBOBJECT_TYPE	The type of a state subobject. Use with D3D12_STATE_SUBOBJECT.
D3D12_STATIC_BORDER_COLOR	Specifies the border color for a static sampler.
D3D12_STENCIL_OP	Identifies the stencil operations that can be performed during depth-stencil testing.
D3D12_TEXTURE_ADDRESS_MODE	Identifies a technique for resolving texture coordinates that are outside of the boundaries of a texture.
D3D12_TEXTURE_BARRIER_FLAGS	
D3D12_TEXTURE_COPY_TYPE	Specifies what type of texture copy is to take place.
D3D12_TEXTURE_LAYOUT	Specifies texture layout options. (D3D12_TEXTURE_LAYOUT)
D3D12_TILE_COPY_FLAGS	Specifies how to copy a tile.
D3D12_TILE_MAPPING_FLAGS	Specifies how to perform a tile-mapping operation.
D3D12_TILE_RANGE_FLAGS	Specifies a range of tile mappings.
D3D12_TILED_RESOURCES_TIER	Identifies the tier level at which tiled resources are supported.

[D3D12_TRI_STATE](#)

TBD

[D3D12_UAV_DIMENSION](#)

Identifies unordered-access view options.

[D3D12_VARIABLE_SHADING_RATE_TIER](#)

Defines constants that specify a shading rate tier (for variable-rate shading, or VRS).

[D3D12_VIEW_INSTANCING_FLAGS](#)

Specifies options for view instancing.

[D3D12_VIEW_INSTANCING_TIER](#)

Indicates the tier level at which view instancing is supported.

[D3D12_WAVE_MMA_TIER](#)

Defines constants that specify a level of support for WaveMMA (wave_matrix) operations.

[D3D12_WRITEBUFFERIMMEDIATE_MODE](#)

Specifies the mode used by a WriteBufferImmediate operation.

Functions

[\[+\] Expand table](#)

[AcquireDirect3D12BufferResource](#)

The IHolographicCameraInterop::AcquireDirect3D12BufferResource function acquires a Direct3D 12 buffer resource.

[AcquireDirect3D12BufferResource](#)

The IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource function acquires a Direct3D 12 buffer resource.

[AcquireDirect3D12BufferResourceWithTimeout](#)

The IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout function acquires a Direct3D 12 buffer resource, with an optional timeout.

[AcquireDirect3D12BufferResourceWithTimeout](#)

The IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout function acquires a Direct3D 12 buffer resource, with an optional timeout.

[AcquireWrappedResources](#)

Acquires D3D11 resources for use with D3D 11on12. Indicates that rendering to the wrapped resources can begin again.

[AddApplicationMessage](#)

Adds a user-defined message to the message queue and sends that message to debug output.

[AddMessage](#)

Adds a debug message to the message queue and sends that message to debug output.

[AddRetrievalFilterEntries](#)

Add storage filters to the top of the retrieval-filter stack.
(ID3D12InfoQueue.AddRetrievalFilterEntries)

[AddStorageFilterEntries](#)

Add storage filters to the top of the storage-filter stack.
(ID3D12InfoQueue.AddStorageFilterEntries)

[AddToObject](#)

Incrementally add to an existing state object. This incurs lower CPU overhead than creating a state object from scratch that is a superset of an existing one.

[AssertResourceState](#)

Checks whether a resource, or subresource, is in a specified state, or not.
(ID3D12DebugCommandList.AssertResourceState)

[AssertResourceState](#)

Validates that the given state matches the state of the subresource, assuming the state of the given subresource is known during recording of a command list (e.g.

[AssertResourceState](#)

Checks whether a resource, or subresource, is in a specified state, or not. (ID3D12DebugCommandQueue.AssertResourceState)
AtomicCopyBufferUINT
Atomically copies a primary data element of type UINT from one resource to another, along with optional dependent resources.
AtomicCopyBufferUINT64
Atomically copies a primary data element of type UINT64 from one resource to another, along with optional dependent resources.
Barrier
Adds a collection of barriers into a graphics command list recording.
BeginEvent
Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (ID3D12CommandQueue.BeginEvent)
BeginEvent
Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (ID3D12GraphicsCommandList.BeginEvent)
BeginQuery
Starts a query running. (ID3D12GraphicsCommandList.BeginQuery)
BeginRenderPass
Marks the beginning of a render pass by binding a set of output resources for the duration of the render pass. These bindings are to one or more render target views (RTVs), and/or to a depth stencil view (DSV).
BuildRaytracingAccelerationStructure
Performs a raytracing acceleration structure build on the GPU and optionally outputs post-build information immediately after the build.
CheckDriverMatchingIdentifier
Reports the compatibility of serialized data, such as a serialized raytracing acceleration structure resulting from a call to CopyRaytracingAccelerationStructure with mode D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE, with the current device/driver.

CheckFeatureSupport	Gets information about the features that are supported by the current graphics driver. (ID3D12Device.CheckFeatureSupport)
ClearDepthStencilView	Clears the depth-stencil resource. (ID3D12GraphicsCommandList.ClearDepthStencilView)
ClearRenderTargetView	Sets all the elements in a render target to one value.
ClearRetrievalFilter	Remove a retrieval filter from the top of the retrieval-filter stack. (ID3D12InfoQueue.ClearRetrievalFilter)
ClearState	Resets the state of a direct command list back to the state it was in when the command list was created. (ID3D12GraphicsCommandList.ClearState)
ClearStorageFilter	Remove a storage filter from the top of the storage-filter stack. (ID3D12InfoQueue.ClearStorageFilter)
ClearStoredMessages	Clear all messages from the message queue. (ID3D12InfoQueue.ClearStoredMessages)
ClearUnorderedAccessViewFloat	Sets all the elements in a unordered access view to the specified float values.
ClearUnorderedAccessViewUint	Sets all the elements in a unordered-access view (UAV) to the specified integer values.
Close	Indicates that recording to the command list has finished. (ID3D12GraphicsCommandList.Close)
CommitDirect3D12Resource	The IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource function commits a Direct3D 12 buffer for presentation on HolographicCamera outputs.

[CommitDirect3D12Resource](#)

Commits a Direct3D 12 buffer for presentation on outputs associated with any [HolographicCamera](#) to which the quad layer is attached.

[CommitDirect3D12ResourceWithDepthData](#)

The

`IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData` function commits a Direct3D 12 buffer for HolographicCamera outputs.

[CopyBufferRegion](#)

Copies a region of a buffer from one resource to another.

[CopyDescriptors](#)

Copies descriptors from a source to a destination. (`ID3D12Device.CopyDescriptors`)

[CopyDescriptorsSimple](#)

Copies descriptors from a source to a destination. (`ID3D12Device.CopyDescriptorsSimple`)

[CopyRaytracingAccelerationStructure](#)

Copies a source acceleration structure to destination memory while applying the specified transformation.

[CopyResource](#)

Copies the entire contents of the source resource to the destination resource.

[CopyTextureRegion](#)

This method uses the GPU to copy texture data between two locations. Both the source and the destination may reference texture data located within either a buffer resource or a texture resource.

[CopyTileMappings](#)

Copies mappings from a source reserved resource to a destination reserved resource.

[CopyTiles](#)

Copies tiles from buffer to tiled resource or vice versa. (`ID3D12GraphicsCommandList.CopyTiles`)

[CreateCommandAllocator](#)

Creates a command allocator object.

[CreateCommandList](#)

Creates a command list.

[CreateCommandList1](#)

Creates a command list in the closed state.

[CreateCommandQueue](#)

Creates a command queue.

[CreateCommandQueue1](#)

Creates a command queue with a creator ID.

[CreateCommandSignature](#)

This method creates a command signature.

[CreateCommittedResource](#)

Creates both a resource and an implicit heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.

[CreateCommittedResource1](#)

Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.
(ID3D12Device4::CreateCommittedResource1)

[CreateCommittedResource2](#)

Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.

[CreateCommittedResource3](#)

Creates a committed resource with an initial layout rather than an initial state.

[CreateComputePipelineState](#)

Creates a compute pipeline state object.

[CreateConstantBufferView](#)

Creates a constant-buffer view for accessing resource data.

CreateDepthStencilView	Creates a depth-stencil view for accessing resource data.
CreateDescriptorHeap	Creates a descriptor heap object.
CreateDirect3D12BackBufferResource	Creates a Direct3D 12 resource for use as a content buffer for the camera.
CreateDirect3D12ContentBufferResource	Creates a Direct3D 12 resource for use as a content buffer for the layer.
CreateDirect3D12HardwareProtectedBackBufferResource	IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource creates a Direct3D 12 resource for use as a content buffer for the camera.
CreateDirect3D12HardwareProtectedContentBufferResource	The IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource function creates a Direct3D 12 resource content buffer for the camera.
CreateFence	Creates a fence object. (ID3D12Device.CreateFence)
CreateFenceFd	TBD
CreateGraphicsPipelineState	Creates a graphics pipeline state object.
CreateHeap	Creates a heap that can be used with placed resources and reserved resources.
CreateHeap1	Creates a heap (optionally for a protected session) that can be used with placed resources and reserved resources.
CreateLifetimeTracker	

Creates a lifetime tracker associated with an application-defined callback; the callback receives notifications when the lifetime of a tracked object is changed.

[CreateMetaCommand](#)

Creates an instance of the specified meta command.

[CreatePipelineLibrary](#)

Creates a cached pipeline library.

[CreatePipelineState](#)

Creates a pipeline state object from a pipeline state stream description.

[CreatePlacedResource](#)

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

[CreatePlacedResource1](#)

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

[CreatePlacedResource2](#)

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

[CreateProtectedResourceSession](#)

Creates an object that represents a session for content protection.

[CreateProtectedResourceSession1](#)

Revises the [ID3D12Device4::CreateProtectedResourceSession](#) method with provision **GUID** that indicates the type of protected resource session.

[CreateQueryHeap](#)

Creates a query heap. A query heap contains an array of queries.

[CreateRenderTargetView](#)

Creates a render-target view for accessing resource data. ([ID3D12Device.CreateRenderTargetView](#))

[CreateReservedResource](#)

	<p>Creates a resource that is reserved, and not yet mapped to any pages in a heap.</p>
CreateReservedResource1	<p>Creates a resource (optionally for a protected session) that is reserved, and not yet mapped to any pages in a heap.</p>
CreateReservedResource2	<p>Creates a resource that is reserved, and not yet mapped to any pages in a heap.</p>
CreateRootSignature	<p>Creates a root signature layout.</p>
CreateSampler	<p>Create a sampler object that encapsulates sampling information for a texture.</p>
CreateSamplerFeedbackUnorderedAccessView	<p>For purposes of sampler feedback, creates a descriptor suitable for binding.</p>
CreateShaderCacheSession	<p>Creates an object that grants access to a shader cache, potentially opening an existing cache or creating a new one.</p>
CreateShaderResourceView	<p>Creates a shader-resource view for accessing data in a resource. (ID3D12Device.CreateShaderResourceView)</p>
CreateSharedHandle	<p>Creates a shared handle to a heap, resource, or fence object.</p>
CreateStateObject	<p>Creates an ID3D12StateObject.</p>
CreateUnorderedAccessView	<p>Creates a view for unordered accessing.</p>
CreateWrappedResource	<p>This method creates D3D11 resources for use with D3D 11on12.</p>

[D3D11On12CreateDevice](#)

Creates a device that uses Direct3D 11 functionality in Direct3D 12, specifying a pre-existing Direct3D 12 device to use for Direct3D 11 interop.

[D3D12_DECODE_FILTER_REDUCTION](#)

[D3D12_DECODE_IS_ANISOTROPIC_FILTER](#)

[D3D12_DECODE_IS_COMPARISON_FILTER](#)

[D3D12_DECODE_MAG_FILTER](#)

[D3D12_DECODE_MIN_FILTER](#)

[D3D12_DECODE_MIP_FILTER](#)

[D3D12_DECODE_SHADER_4_COMPONENT_MAPPING](#)

[D3D12_ENCODE_ANISOTROPIC_FILTER](#)

[D3D12_ENCODE_BASIC_FILTER](#)

[D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING](#)

[D3D12_GET_COARSE_SHADING_RATE_X_AXIS](#)

[D3D12_GET_COARSE_SHADING_RATE_Y_AXIS](#)

[D3D12_MAKE_COARSE_SHADING_RATE](#)

[D3D12CreateDevice](#)

Creates a device that represents the display adapter. (D3D12CreateDevice)

[D3D12CreateRootSignatureDeserializer](#)

Deserializes a root signature so you can determine the layout definition (D3D12_ROOT_SIGNATURE_DESC).
D3D12CreateVersionedRootSignatureDeserializer
Generates an interface that can return the deserialized data structure, via GetUnconvertedRootSignatureDesc.
D3D12EnableExperimentalFeatures
Enables a list of experimental features.
D3D12GetDebugInterface
Gets a debug interface.
D3D12GetInterface
Selects an SDK version at runtime when the system is in Windows Developer Mode.
D3D12SerializeRootSignature
Serializes a root signature version 1.0 that can be passed to ID3D12Device::CreateRootSignature.
D3D12SerializeVersionedRootSignature
Serializes a root signature of any version that can be passed to ID3D12Device::CreateRootSignature.
DestroyOwnedObject
Destroys a lifetime-tracked object.
DisableDebugLayer
Disables the debug layer.
DiscardResource
Discards a resource.
Dispatch
Executes a compute shader on a thread group.
DispatchMesh
DispatchRays

	Launch the threads of a ray generation shader.
DrawIndexedInstanced	Draws indexed, instanced primitives.
DrawInstanced	Draws non-indexed, instanced primitives.
EmitRaytracingAccelerationStructurePostbuildInfo	Emits post-build properties for a set of acceleration structures. This enables applications to know the output resource requirements for performing acceleration structure operations via <code>ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure</code> .
EnableDebugLayer	Enables the debug layer. (<code>ID3D12Debug.EnableDebugLayer</code>)
EnableDebugLayer	Enables the debug layer. (<code>ID3D12Debug1.EnableDebugLayer</code>)
EnableShaderInstrumentation	This method enables tools such as PIX to instrument shaders.
EndEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (<code>ID3D12CommandQueue.EndEvent</code>)
EndEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (<code>ID3D12GraphicsCommandList.EndEvent</code>)
EndQuery	Ends a running query.
EndRenderPass	Marks the ending of a render pass.
EnqueueMakeResident	Asynchronously makes objects resident for the device.

EnumerateMetaCommandParameters	Queries reflection metadata about the parameters of the specified meta command.
EnumerateMetaCommands	Queries reflection metadata about available meta commands.
Evict	Enables the page-out of data, which precludes GPU access of that data.
ExecuteBundle	Executes a bundle.
ExecuteCommandLists	Submits an array of command lists for execution.
ExecuteIndirect	Apps perform indirect draws/dispatches using the ExecuteIndirect method.
ExecuteMetaCommand	Records the execution (or invocation) of the specified meta command into a graphics command list.
FindValue	Looks up an entry in the cache whose key exactly matches the provided key.
GetAdapterLuid	Gets a locally unique identifier for the current device (adapter).
GetAutoBreadcrumbsOutput	Retrieves the Device Removed Extended Data (DRED) auto-breadcrumbs output after device removal.
GetAutoBreadcrumbsOutput1	
GetBaseClass	Gets an ID3D12ShaderReflectionType Interface interface containing the variable base class type.

GetBitwiseInstructionCount	Gets the number of bitwise instructions. (<code>ID3D12ShaderReflection.GetBitwiseInstructionCount</code>)
GetBreakOnCategory	Get a message category to break on when a message with that category passes through the storage filter. (<code>ID3D12InfoQueue.GetBreakOnCategory</code>)
GetBreakOnID	Get a message identifier to break on when a message with that identifier passes through the storage filter. (<code>ID3D12InfoQueue.GetBreakOnID</code>)
GetBreakOnSeverity	Get a message severity level to break on when a message with that severity level passes through the storage filter. (<code>ID3D12InfoQueue.GetBreakOnSeverity</code>)
GetBuffer	Returns the <code>ID3D12ShaderReflectionConstantBuffer</code> of the present <code>ID3D12ShaderReflectionVariable</code> .
GetCachedBlob	Gets the cached blob representing the pipeline state.
GetClockCalibration	This method samples the CPU and GPU timestamp counters at the same moment in time.
GetCompletedValue	Gets the current value of the fence. (<code>ID3D12Fence.GetCompletedValue</code>)
GetConstantBufferByIndex	The <code>ID3D12FunctionReflection::GetConstantBufferByIndex</code> method (<code>d3d12shader.h</code>) gets a constant buffer by index for a function.
GetConstantBufferByIndex	Gets a constant buffer by index.
GetConstantBufferByName	Gets a constant buffer by name for a function. (<code>ID3D12FunctionReflection.GetConstantBufferByName</code>)

[GetConstantBufferByName](#)

Gets a constant buffer by name.

[GetConversionInstructionCount](#)

Gets the number of conversion instructions.

(ID3D12ShaderReflection.GetConversionInstructionCount)

[GetCopyableFootprints](#)

Gets a resource layout that can be copied. Helps the app fill-in

D3D12_PLACED_SUBRESOURCE_FOOTPRINT and D3D12_SUBRESOURCE_FOOTPRINT when
suballocating space in upload heaps.

[GetCopyableFootprints1](#)

Gets a resource layout that can be copied. Helps your app fill in

D3D12_PLACED_SUBRESOURCE_FOOTPRINT and D3D12_SUBRESOURCE_FOOTPRINT when
suballocating space in upload heaps.

[GetCPUDescriptorHandleForHeapStart](#)

Gets the CPU descriptor handle that represents the start of the heap.

[GetCreationFlags](#)

Gets the flags used to create the fence represented by the current instance.

[GetCurrentResourceAndCommandQueue](#)

[GetCustomHeapProperties](#)

Divulges the equivalent custom heap properties that are used for non-custom heap types, based
on the adapter's architectural properties.

[GetD3D12Device](#)

Retrieves the [Direct3D 12 device](#) being interoperated with.

[GetDebugParameter](#)

Gets optional Command List Debug Layer settings.

[GetDebugParameter](#)

Gets optional device-wide Debug Layer settings.

[GetDesc](#)

Gets the description of the command queue.

[GetDesc](#)

Gets the descriptor heap description.

[GetDesc](#)

Gets the heap description.

[GetDesc](#)

Retrieves a description of the protected resource session.

(ID3D12ProtectedResourceSession.GetDesc)

[GetDesc](#)

Gets the resource description.

[GetDesc](#)

Retrieves the description used to create the cache session.

[GetDesc](#)

Fills the parameter descriptor structure for the function's parameter.

(ID3D12FunctionParameterReflection.GetDesc)

[GetDesc](#)

Fills the function descriptor structure for the function. (ID3D12FunctionReflection.GetDesc)

[GetDesc](#)

Fills the library descriptor structure for the library reflection. (ID3D12LibraryReflection.GetDesc)

[GetDesc](#)

Gets a shader description.

[GetDesc](#)

Gets a constant-buffer description.

[GetDesc](#)

Gets the description of a shader-reflection-variable type.

GetDesc
Gets a shader-variable description.
GetDesc1
Retrieves a description of the protected resource session. (ID3D12ProtectedResourceSession1::GetDesc1)
GetDesc1
GetDescriptorHandleIncrementSize
Gets the size of the handle increment for the given type of descriptor heap. This value is typically used to increment a handle into a descriptor array by the correct amount.
GetDevice
Gets a pointer to the device that created this interface.
GetDeviceRemovedReason
Gets the reason that the device was removed.
GetDeviceState
GetFeatureMask
Returns the debug feature flags that have been set on a command list.
GetFeatureMask
Gets a bit field of flags that indicates which debug features are on or off.
GetFunctionByIndex
The ID3D12LibraryReflection::GetFunctionByIndex method (d3d12shader.h) gets the function reflector.
GetFunctionParameter
Gets the function parameter reflector. (ID3D12FunctionReflection.GetFunctionParameter)
GetGPUDescriptorHandleForHeapStart
Gets the GPU descriptor handle that represents the start of the heap.

[GetGPUVirtualAddress](#)

This method returns the GPU virtual address of a buffer resource.

[GetGSIInputPrimitive](#)

Gets the geometry-shader input-primitive description.

(ID3D12ShaderReflection.GetGSIInputPrimitive)

[GetHeapProperties](#)

Retrieves the properties of the resource heap, for placed and committed resources.

[GetInputParameterDesc](#)

Gets an input-parameter description for a shader.

[GetInterfaceByIndex](#)

Gets an interface by index.

[GetInterfaceSlot](#)

Gets the corresponding interface slot for a variable that represents an interface pointer.

(ID3D12ShaderReflectionVariable.GetInterfaceSlot)

[GetLUID](#)

[GetMemberTypeByIndex](#)

Gets a shader-reflection-variable type by index.

[GetMemberTypeByName](#)

Gets a shader-reflection-variable type by name.

[GetMemberTypeName](#)

Gets a shader-reflection-variable type.

[GetMessage](#)

Get a message from the message queue. (ID3D12InfoQueue.GetMessage)

[GetMessageCountLimit](#)

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

(ID3D12InfoQueue.GetMessageCountLimit)

[GetMinFeatureLevel](#)

Gets the minimum feature level. (`ID3D12ShaderReflection.GetMinFeatureLevel`)

[GetMovcInstructionCount](#)

Gets the number of Movc instructions. (`ID3D12ShaderReflection.GetMovcInstructionCount`)

[GetMovInstructionCount](#)

Gets the number of Mov instructions. (`ID3D12ShaderReflection.GetMovInstructionCount`)

[GetMuteDebugOutput](#)

Get a boolean that determines if debug output is on or off.

[GetNodeCount](#)

Reports the number of physical adapters (nodes) that are associated with this device.

[GetNumInterfaces](#)

Gets the number of interfaces. (`ID3D12ShaderReflectionType.GetNumInterfaces`)

[GetNumInterfaceSlots](#)

Gets the number of interface slots in a shader. (`ID3D12ShaderReflection.GetNumInterfaceSlots`)

[GetNumMessagesAllowedByStorageFilter](#)

Get the number of messages that were allowed to pass through a storage filter.
(`ID3D12InfoQueue.GetNumMessagesAllowedByStorageFilter`)

[GetNumMessagesDeniedByStorageFilter](#)

Get the number of messages that were denied passage through a storage filter.
(`ID3D12InfoQueue.GetNumMessagesDeniedByStorageFilter`)

[GetNumMessagesDiscardedByMessageCountLimit](#)

Get the number of messages that were discarded due to the message count limit.
(`ID3D12InfoQueue.GetNumMessagesDiscardedByMessageCountLimit`)

[GetNumStoredMessages](#)

Get the number of messages currently stored in the message queue.
(`ID3D12InfoQueue.GetNumStoredMessages`)

[GetNumStoredMessagesAllowedByRetrievalFilter](#)

Get the number of messages that are able to pass through a retrieval filter.
(ID3D12InfoQueue.GetNumStoredMessagesAllowedByRetrievalFilter)

[GetOutputParameterDesc](#)

Gets an output-parameter description for a shader.

[GetPageFaultAllocationOutput](#)

Retrieves the Device Removed Extended Data (DRED) page fault data.

[GetPageFaultAllocationOutput1](#)[GetPageFaultAllocationOutput2](#)[GetPatchConstantParameterDesc](#)

Gets a patch-constant parameter description for a shader.

[GetPipelineStackSize](#)

Gets the current pipeline stack size.

[GetPrivateData](#)

Gets application-defined data from a device object.

[GetProtectedResourceSession](#)[GetProtectedResourceSession](#)[GetRaytracingAccelerationStructurePrebuildInfo](#)

Query the driver for resource requirements to build an acceleration structure.

[GetRequiredParameterResourceSize](#)

Retrieves the amount of memory required for the specified runtime parameter resource for a meta command, for the specified stage.

[GetRequiresFlags](#)

	Gets a group of flags that indicates the requirements of a shader. (ID3D12ShaderReflection.GetRequiresFlags)
GetResourceAllocationInfo	Gets the size and alignment of memory required for a collection of resources on this adapter.
GetResourceAllocationInfo1	Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. (ID3D12Device4::GetResourceAllocationInfo1)
GetResourceAllocationInfo2	Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. (ID3D12Device8::GetResourceAllocationInfo2)
GetResourceBindingDesc	Gets a description of how a resource is bound to a function. (ID3D12FunctionReflection.GetResourceBindingDesc)
GetResourceBindingDesc	Gets a description of how a resource is bound to a shader. (ID3D12ShaderReflection.GetResourceBindingDesc)
GetResourceBindingDescByName	Gets a description of how a resource is bound to a function. (ID3D12FunctionReflection.GetResourceBindingDescByName)
GetResourceBindingDescByName	Gets a description of how a resource is bound to a shader. (ID3D12ShaderReflection.GetResourceBindingDescByName)
GetResourceTiling	Gets info about how a tiled resource is broken into tiles. (ID3D12Device.GetResourceTiling)
GetRetrievalFilter	Get the retrieval filter at the top of the retrieval-filter stack. (ID3D12InfoQueue.GetRetrievalFilter)
GetRetrievalFilterStackSize	Get the size of the retrieval-filter stack in bytes. (ID3D12InfoQueue.GetRetrievalFilterStackSize)

GetRootSignatureDesc
Gets the layout of the root signature.
GetRootSignatureDescAtVersion
Converts root signature description structures to a requested version.
GetSerializedSize
Returns the amount of memory required to serialize the current contents of the database.
GetSessionStatus
Gets the status of the protected session.
GetShaderIdentifier
Retrieves the unique identifier for a shader that can be used in a shader record.
GetShaderStackSize
Gets the amount of stack memory required to invoke a raytracing shader in HLSL.
GetStatusFence
Retrieves the fence for the protected session. From the fence, you can retrieve the current uniqueness validity value (using <code>ID3D12Fence::GetCompletedValue</code>), and add monitors for changes to its value. This is a read-only fence.
GetStorageFilter
Get the storage filter at the top of the storage-filter stack. (<code>ID3D12InfoQueue.GetStorageFilter</code>)
GetStorageFilterStackSize
Get the size of the storage-filter stack in bytes. (<code>ID3D12InfoQueue.GetStorageFilterStackSize</code>)
GetSubType
Gets the base class of a class. (<code>ID3D12ShaderReflectionType.GetSubType</code>)
GetSwapChainObject
GetThreadGroupSize
Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid. (<code>ID3D12ShaderReflection.GetThreadGroupSize</code>)

[GetTimestampFrequency](#)

This method is used to determine the rate at which the GPU timestamp counter increments.

[GetType](#)

Gets the type of the command list, such as direct, bundle, compute, or copy.

[GetType](#)

Gets a shader-variable type.

[GetUnconvertedRootSignatureDesc](#)

Gets the layout of the root signature, without converting between root signature versions.

[GetVariableByIndex](#)

Gets a shader-reflection variable by index.

[GetVariableByName](#)

Gets a variable by name. (`ID3D12FunctionReflection.GetVariableByName`)

[GetVariableByName](#)

Gets a variable by name. (`ID3D12ShaderReflection.GetVariableByName`)

[GetVariableByName](#)

Gets a shader-reflection variable by name.

[IASetIndexBuffer](#)

Sets the view for the index buffer.

[IASetPrimitiveTopology](#)

Bind information about the primitive type, and data order that describes input data for the input assembler stage. (`ID3D12GraphicsCommandList.IASetPrimitiveTopology`)

[IASetVertexBuffers](#)

Sets a CPU descriptor handle for the vertex buffers.

[ImplementsInterface](#)

Indicates whether a class type implements an interface.
(`ID3D12ShaderReflectionType.ImplementsInterface`)

InitializeMetaCommand
Initializes the specified meta command.
InsertImplicitSync
IsEqual
Indicates whether two ID3D12ShaderReflectionType Interface pointers have the same underlying type.
IsOfType
Indicates whether a variable is of the specified type. (ID3D12ShaderReflectionType.IsOfType)
IsSampleFrequencyShader
Indicates whether a shader is a sample frequency shader. (ID3D12ShaderReflection.IsSampleFrequencyShader)
LifetimeStateUpdated
Called when the lifetime state of a lifetime-tracked object changes.
LoadComputePipeline
Retrieves the requested PSO from the library. The input desc is matched against the data in the current library database, and remembered in order to prevent duplication of PSO contents.
LoadGraphicsPipeline
Retrieves the requested PSO from the library.
LoadPipeline
Retrieves the requested PSO from the library. The pipeline stream description is matched against the library database, and remembered in order to prevent duplication of PSO contents.
MakeResident
Makes objects resident for the device.
Map
Gets a CPU pointer to the specified subresource in the resource, but may not disclose the pointer value to applications. Map also invalidates the CPU cache, when necessary, so that CPU reads to this address reflect any modifications made by the GPU.

[OMSetBlendFactor](#)

Sets the blend factor that modulate values for a pixel shader, render target, or both.

[OMSetDepthBounds](#)

This method enables you to change the depth bounds dynamically.

[OMSetRenderTarget](#)s

Sets CPU descriptor handles for the render targets and depth stencil.

[OMSetStencilRef](#)

Sets the reference value for depth stencil tests.

[OpenExistingHeapFromAddress](#)

Creates a special-purpose diagnostic heap in system memory from an address. The created heap can persist even in the event of a GPU-fault or device-removed scenario.

[OpenExistingHeapFromFileMapping](#)

Creates a special-purpose diagnostic heap in system memory from a file mapping object. The created heap can persist even in the event of a GPU-fault or device-removed scenario.

[OpenSharedHandle](#)

Opens a handle for shared resources, shared heaps, and shared fences, by using HANDLE and REFIID.

[OpenSharedHandleByName](#)

Opens a handle for shared resources, shared heaps, and shared fences, by using Name and Access.

[PFN_D3D12_CREATE_DEVICE](#)

[PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER](#)

[PFN_D3D12_CREATE_VERSIONED_ROOT_SIGNATURE_DESERIALIZER](#)

[PFN_D3D12_GET_DEBUG_INTERFACE](#)

[PFN_D3D12_GET_INTERFACE](#)

PFN_D3D12_SERIALIZE_ROOT_SIGNATURE
PFN_D3D12_SERIALIZE_VERSIONED_ROOT_SIGNATURE
PopRetrievalFilter
Pop a retrieval filter from the top of the retrieval-filter stack. (<code>ID3D12InfoQueue.PopRetrievalFilter</code>)
PopStorageFilter
Pop a storage filter from the top of the storage-filter stack. (<code>ID3D12InfoQueue.PopStorageFilter</code>)
Present
Shares a resource (or subresource) between the D3D layers and diagnostics tools.
PushCopyOfRetrievalFilter
Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack. (<code>ID3D12InfoQueue.PushCopyOfRetrievalFilter</code>)
PushCopyOfStorageFilter
Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack. (<code>ID3D12InfoQueue.PushCopyOfStorageFilter</code>)
PushEmptyRetrievalFilter
Push an empty retrieval filter onto the retrieval-filter stack. (<code>ID3D12InfoQueue.PushEmptyRetrievalFilter</code>)
PushEmptyStorageFilter
Push an empty storage filter onto the storage-filter stack. (<code>ID3D12InfoQueue.PushEmptyStorageFilter</code>)
PushRetrievalFilter
Push a retrieval filter onto the retrieval-filter stack. (<code>ID3D12InfoQueue.PushRetrievalFilter</code>)
PushStorageFilter
Push a storage filter onto the storage-filter stack. (<code>ID3D12InfoQueue.PushStorageFilter</code>)
ReadFromSubresource

Uses the CPU to copy data from a subresource, enabling the CPU to read the contents of most textures with undefined layouts.

[ReleaseWrappedResources](#)

Releases D3D11 resources that were wrapped for D3D 11on12.

[RemoveDevice](#)

You can call **RemoveDevice** to indicate to the Direct3D 12 runtime that the GPU device encountered a problem, and can no longer be used.

[ReportLiveDeviceObjects](#)

Reports information about a device object's lifetime.

[ReportLiveDeviceObjects](#)

Specifies the amount of information to report on a device object's lifetime.

[Reset](#)

Indicates to re-use the memory that is associated with the command allocator.

[Reset](#)

Resets a command list back to its initial state as if a new command list was just created.
(ID3D12GraphicsCommandList.Reset)

[ResolveQueryData](#)

Extracts data from a query. ResolveQueryData works with all heap types (default, upload, and readback). ResolveQueryData works with all heap types (default, upload, and readback).

[ResolveSubresource](#)

Copy a multi-sampled resource into a non-multi-sampled resource.

[ResolveSubresourceRegion](#)

Copy a region of a multisampled or compressed resource into a non-multisampled or non-compressed resource.

[ResourceBarrier](#)

Notifies the driver that it needs to synchronize multiple accesses to resources.
(ID3D12GraphicsCommandList.ResourceBarrier)

[ReturnUnderlyingResource](#)

With this method, you can return a Direct3D 11 resource object to Direct3D11On12, and indicate when the resource will be ready to consume.

[RSSetScissorRects](#)

Binds an array of scissor rectangles to the rasterizer stage.

[RSSetShadingRate](#)

The ID3D12GraphicsCommandList5::RSSetShadingRate method (d3d12.h) sets the base shading rate, and combiners, for variable-rate shading (VRS).

[RSSetShadingRateImage](#)

The ID3D12GraphicsCommandList5::RSSetShadingRateImage method (d3d12.h) sets the screen-space shading-rate image for variable-rate shading (VRS).

[RSSetViewports](#)

Bind an array of viewports to the rasterizer stage of the pipeline.
(ID3D12GraphicsCommandList.RSSetViewports)

[Serialize](#)

Writes the contents of the library to the provided memory, to be provided back to the runtime at a later time.

[SetAutoBreadcrumbsEnablement](#)

Configures the enablement settings for Device Removed Extended Data (DRED) auto-breadcrumbs.

[SetBackgroundProcessingMode](#)

Sets the mode for driver background processing optimizations.

[SetBreadcrumbContextEnablement](#)

[SetBreakOnCategory](#)

Set a message category to break on when a message with that category passes through the storage filter. (ID3D12InfoQueue.SetBreakOnCategory)

[SetBreakOnID](#)

Set a message identifier to break on when a message with that identifier passes through the storage filter. (ID3D12InfoQueue.SetBreakOnID)

SetBreakOnSeverity	Set a message severity level to break on when a message with that severity level passes through the storage filter. (<code>ID3D12InfoQueue::SetBreakOnSeverity</code>)
SetComputeRoot32BitConstant	Sets a constant in the compute root signature.
SetComputeRoot32BitConstants	Sets a group of constants in the compute root signature.
SetComputeRootConstantBufferView	Sets a CPU descriptor handle for the constant buffer in the compute root signature.
SetComputeRootDescriptorTable	Sets a descriptor table into the compute root signature.
SetComputeRootShaderResourceView	Sets a CPU descriptor handle for the shader resource in the compute root signature.
SetComputeRootSignature	Sets the layout of the compute root signature.
SetComputeRootUnorderedAccessView	Sets a CPU descriptor handle for the unordered-access-view resource in the compute root signature.
SetDebugParameter	Modifies optional Debug Layer settings of a command list.
SetDebugParameter	Modifies the D3D12 optional device-wide Debug Layer settings.
SetDeleteOnDestroy	When all cache session objects corresponding to a given cache are destroyed, the cache is cleared.
SetDescriptorHeaps	

Changes the currently bound descriptor heaps that are associated with a command list.

[SetEnableAutoName](#)

Configures the auto-naming of objects.

[SetEnableGPUBasedValidation](#)

This method enables or disables GPU-Based Validation (GBV) before creating a device with the debug layer enabled.

[SetEnableGPUBasedValidation](#)

This method enables or disables GPU-based validation (GBV) before creating a device with the debug layer enabled.

[SetEnableSynchronizedCommandQueueValidation](#)

Enables or disables dependent command queue synchronization when using a D3D12 device with the debug layer enabled.

[SetEnableSynchronizedCommandQueueValidation](#)

Enables or disables dependent command queue synchronization when using a Direct3D 12 device with the debug layer enabled.

[SetEventOnCompletion](#)

Specifies an event that should be fired when the fence reaches a certain value.

(ID3D12Fence.SetEventOnCompletion)

[SetEventOnMultipleFenceCompletion](#)

Specifies an event that should be fired when one or more of a collection of fences reach specific values.

[SetFeatureMask](#)

Turns the debug features for a command list on or off.

[SetFeatureMask](#)

Set a bit field of flags that will turn debug features on and off.

(ID3D12DebugDevice.SetFeatureMask)

[SetForceLegacyBarrierValidation](#)

TBD

[SetGPUBasedValidationFlags](#)

This method configures the level of GPU-based validation that the debug device is to perform at runtime. (ID3D12Debug2.SetGPUBasedValidationFlags)

[SetGPUBasedValidationFlags](#)

This method configures the level of GPU-based validation that the debug device is to perform at runtime. (ID3D12Debug3.SetGPUBasedValidationFlags)

[SetGraphicsRoot32BitConstant](#)

Sets a constant in the graphics root signature.

[SetGraphicsRoot32BitConstants](#)

Sets a group of constants in the graphics root signature.

[SetGraphicsRootConstantBufferView](#)

Sets a CPU descriptor handle for the constant buffer in the graphics root signature.

[SetGraphicsRootDescriptorTable](#)

Sets a descriptor table into the graphics root signature.

[SetGraphicsRootShaderResourceView](#)

Sets a CPU descriptor handle for the shader resource in the graphics root signature.

[SetGraphicsRootSignature](#)

Sets the layout of the graphics root signature.

[SetGraphicsRootUnorderedAccessView](#)

Sets a CPU descriptor handle for the unordered-access-view resource in the graphics root signature.

[SetMarker](#)

Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (ID3D12CommandQueue.SetMarker)

[SetMarker](#)

Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (ID3D12GraphicsCommandList.SetMarker)

SetMessageCountLimit	Set the maximum number of messages that can be added to the message queue. (ID3D12InfoQueue.SetMessageCountLimit)
SetMuteDebugOutput	Set a boolean that turns the debug output on or off. (ID3D12InfoQueue.SetMuteDebugOutput)
SetName	Associates a name with the device object. This name is for use in debug diagnostics and tools.
SetPageFaultEnablement	Configures the enablement settings for Device Removed Extended Data (DRED) page fault reporting.
SetPipelineStackSize	Set the current pipeline stack size.
SetPipelineState	Sets all shaders and programs most of the fixed-function state of the graphics processing unit (GPU) pipeline.
SetPipelineState1	Sets a state object on the command list.
SetPredication	Sets a rendering predicate.
SetPrivateData	Sets application-defined data to a device object and associates that data with an application-defined GUID.
SetPrivateDataInterface	Associates an IUnknown-derived interface with the device object, and associates that interface with an application-defined GUID.
SetProtectedResourceSession	Specifies whether or not protected resources can be accessed by subsequent commands in the command list.

SetResidencyPriority
This method sets residency priorities of a specified list of objects.
SetSamplePositions
This method configures the sample positions used by subsequent draw, copy, resolve, and similar operations.
SetSDKVersion
Configures the SDK version to use.
SetStablePowerState
A development-time aid for certain types of profiling and experimental prototyping.
SetViewInstanceMask
Set a mask that controls which view instances are enabled for subsequent draws.
SetWatsonDumpEnablement
Configures the enablement settings for Device Removed Extended Data (DRED) Watson dump creation.
ShaderCacheControl
Modifies the behavior of caches used internally by Direct3D or by the driver.
ShaderInstrumentationEnabled
Determines whether shader instrumentation is enabled.
SharedFenceSignal
Signals a shared fence between the D3D layers and diagnostics tools.
ShareWithHost
TBD
Signal
Updates a fence to a specified value.
Signal
Sets the fence to the specified value.

SOSetTargets
Sets the stream output buffer views.
StorePipeline
Adds the input PSO to an internal database with the corresponding name.
StoreValue
Adds an entry to the cache.
UnacquireDirect3D12BufferResource
The IHolographicCameraInterop::UnacquireDirect3D12BufferResource function un-acquires a Direct3D 12 buffer resource.
UnacquireDirect3D12BufferResource
The IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource function un-acquires a Direct3D 12 buffer resource.
Unmap
Invalidate the CPU pointer to the specified subresource in the resource.
UnwrapUnderlyingResource
Unwraps a Direct3D 11 resource object, and retrieves it as a Direct3D 12 resource object.
UpdateTileMappings
Updates mappings of tile locations in reserved resources to memory locations in a resource heap.
Wait
Queues a GPU-side wait, and returns immediately. A GPU-side wait is where the GPU waits until the specified fence reaches or exceeds the specified value.
WriteBufferImmediate
Writes a number of 32-bit immediate values to the specified buffer locations directly from the command stream. (ID3D12GraphicsCommandList2::WriteBufferImmediate)
WriteToSubresource
Uses the CPU to copy data into a subresource, enabling the CPU to modify the contents of most textures with undefined layouts.

Interfaces

 Expand table

ID3D11On12Device
Handles the creation, wrapping, and releasing of D3D11 resources for Direct3D11on12.
ID3D11On12Device1
Enables better interoperability with a component that might be handed a Direct3D 11 device, but which wants to leverage Direct3D 12 instead.
ID3D11On12Device2
Enables you to take resources created through the Direct3D 11 APIs, and use them in Direct3D 12.
ID3D12CommandAllocator
Represents the allocations of storage for graphics processing unit (GPU) commands.
ID3D12CommandList
An interface from which ID3D12GraphicsCommandList inherits from. It represents an ordered set of commands that the GPU executes, while allowing for extension to support other command lists than just those for graphics (such as compute and copy).
ID3D12CommandQueue
Provides methods for submitting command lists, synchronizing command list execution, instrumenting the command queue, and updating resource tile mappings.
ID3D12CommandSignature
A command signature object enables apps to specify indirect drawing, including the buffer format, command type and resource bindings to be used.
ID3D12Debug
An interface used to turn on the debug layer.
ID3D12Debug1
Adds GPU-Based Validation and Dependent Command Queue Synchronization to the debug layer.
ID3D12Debug2
Adds configurable levels of GPU-based validation to the debug layer. (ID3D12Debug2)

ID3D12Debug3
Adds configurable levels of GPU-based validation to the debug layer. (ID3D12Debug3)
ID3D12Debug4
Adds the ability to disable the debug layer.
ID3D12Debug5
Adds to the debug layer the ability to configure the auto-naming of objects.
ID3D12Debug6
A debug interface controls debug settings.
ID3D12DebugCommandList
Provides methods to monitor and debug a command list.
ID3D12DebugCommandList1
This interface enables modification of additional command list debug layer settings.
ID3D12DebugCommandQueue
Provides methods to monitor and debug a command queue.
ID3D12DebugDevice
This interface represents a graphics device for debugging.
ID3D12DebugDevice1
Specifies device-wide debug layer settings.
ID3D12DescriptorHeap
A descriptor heap is a collection of contiguous allocations of descriptors, one allocation for every descriptor.
ID3D12Device
Represents a virtual adapter; it is used to create command allocators, command lists, command queues, fences, resources, pipeline state objects, heaps, root signatures, samplers, and many resource views.
ID3D12Device1

Represents a virtual adapter, and expands on the range of methods provided by ID3D12Device.

[ID3D12Device10](#)

TBD

[ID3D12Device2](#)

Represents a virtual adapter. This interface extends ID3D12Device1 to create pipeline state objects from pipeline state stream descriptions.

[ID3D12Device3](#)

Represents a virtual adapter. This interface extends ID3D12Device2 to support the creation of special-purpose diagnostic heaps in system memory that persist even in the event of a GPU-fault or device-removed scenario.

[ID3D12Device4](#)

Represents a virtual adapter. This interface extends [ID3D12Device3](#).

[ID3D12Device5](#)

Represents a virtual adapter. This interface extends [ID3D12Device4](#).

[ID3D12Device6](#)

Represents a virtual adapter. This interface extends [ID3D12Device5](#).

[ID3D12Device7](#)

Represents a virtual adapter. This interface extends [ID3D12Device6](#).

[ID3D12Device8](#)

Represents a virtual adapter. This interface extends [ID3D12Device7](#).

[ID3D12Device9](#)

Represents a virtual adapter. This interface extends [ID3D12Device8](#) to add methods to manage shader caches.

[ID3D12DeviceChild](#)

An interface from which other core interfaces inherit from, including (but not limited to) ID3D12PipelineLibrary, ID3D12CommandList, ID3D12Pageable, and ID3D12RootSignature. It provides a method to get back to the device object it was created against.

[ID3D12DeviceRemovedExtendedData](#)

	Provides runtime access to Device Removed Extended Data (DRED) data.
	ID3D12DeviceRemovedExtendedData1
	ID3D12DeviceRemovedExtendedData2
	ID3D12DeviceRemovedExtendedDataSettings
	This interface controls Device Removed Extended Data (DRED) settings.
	ID3D12DeviceRemovedExtendedDataSettings1
	ID3D12Fence
	Represents a fence, an object used for synchronization of the CPU and one or more GPUs. (ID3D12Fence)
	ID3D12Fence1
	Represents a fence. This interface extends ID3D12Fence, and supports the retrieval of the flags used to create the original fence.
	ID3D12FunctionParameterReflection
	A function-parameter-reflection interface accesses function-parameter info. (ID3D12FunctionParameterReflection)
	ID3D12FunctionReflection
	A function-reflection interface accesses function info. (ID3D12FunctionReflection)
	ID3D12GraphicsCommandList
	Encapsulates a list of graphics commands for rendering. Includes APIs for instrumenting the command list execution, and for setting and clearing the pipeline state.
	ID3D12GraphicsCommandList1
	Encapsulates a list of graphics commands for rendering, extending the interface to support programmable sample positions, atomic copies for implementing late-latch techniques, and optional depth-bounds testing.
	ID3D12GraphicsCommandList2
	Encapsulates a list of graphics commands for rendering, extending the interface to support writing immediate values directly to a buffer.

[ID3D12GraphicsCommandList3](#)

Encapsulates a list of graphics commands for rendering.

[ID3D12GraphicsCommandList4](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support ray tracing and render passes.

[ID3D12GraphicsCommandList5](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support variable-rate shading (VRS).

[ID3D12GraphicsCommandList6](#)

[ID3D12GraphicsCommandList7](#)

TBD

[ID3D12Heap](#)

A heap is an abstraction of contiguous memory allocation, used to manage physical memory. This heap can be used with ID3D12Resource objects to support placed resources or reserved resources.

[ID3D12Heap1](#)

[ID3D12InfoQueue](#)

An information-queue interface stores, retrieves, and filters debug messages. The queue consists of a message queue, an optional storage filter stack, and a optional retrieval filter stack.
(ID3D12InfoQueue)

[ID3D12LibraryReflection](#)

A library-reflection interface accesses library info. (ID3D12LibraryReflection)

[ID3D12LifetimeOwner](#)

Represents an application-defined callback used for being notified of lifetime changes of an object.

[ID3D12LifetimeTracker](#)

Represents facilities for controlling the lifetime a lifetime-tracked object.

[ID3D12MetaCommand](#)

Represents a meta command. A meta command is a Direct3D 12 object representing an algorithm that is accelerated by independent hardware vendors (IHVs). It's an opaque reference to a command generator that is implemented by the driver.

[ID3D12Object](#)

An interface from which ID3D12Device and ID3D12DeviceChild inherit from. It provides methods to associate private data and annotate object names.

[ID3D12Pageable](#)

An interface from which many other core interfaces inherit from. It indicates that the object type encapsulates some amount of GPU-accessible memory; but does not strongly indicate whether the application can manipulate the object's residency.

[ID3D12PipelineLibrary](#)

Manages a pipeline library, in particular loading and retrieving individual PSOs.

[ID3D12PipelineLibrary1](#)

Manages a pipeline library. This interface extends ID3D12PipelineLibrary to load PSOs from a pipeline state stream description.

[ID3D12PipelineState](#)

Represents the state of all currently set shaders as well as certain fixed function state objects.

[ID3D12ProtectedResourceSession](#)

Monitors the validity of a protected resource session. (ID3D12ProtectedResourceSession)

[ID3D12ProtectedResourceSession1](#)

Monitors the validity of a protected resource session. (ID3D12ProtectedResourceSession1)

[ID3D12ProtectedSession](#)

Offers base functionality that allows for a consistent way to monitor the validity of a session across the different types of sessions.

[ID3D12QueryHeap](#)

Manages a query heap. A query heap holds an array of queries, referenced by indexes.

[ID3D12Resource](#)

Encapsulates a generalized ability of the CPU and GPU to read and write to physical memory, or heaps. It contains abstractions for organizing and manipulating simple arrays of data as well as multidimensional data optimized for shader sampling.

[ID3D12Resource1](#)

[ID3D12Resource2](#)

[ID3D12RootSignature](#)

The root signature defines what resources are bound to the graphics pipeline. A root signature is configured by the app and links command lists to the resources the shaders require. Currently, there is one graphics and one compute root signature per app.

[ID3D12RootSignatureDeserializer](#)

Contains a method to return the deserialized D3D12_ROOT_SIGNATURE_DESC data structure, of a serialized root signature version 1.0.

[ID3D12SDKConfiguration](#)

Provides SDK configuration methods.

[ID3D12ShaderCacheSession](#)

Represents a shader cache session.

[ID3D12ShaderReflection](#)

A shader-reflection interface accesses shader information. ([ID3D12ShaderReflection](#))

[ID3D12ShaderReflectionConstantBuffer](#)

This shader-reflection interface provides access to a constant buffer.
([ID3D12ShaderReflectionConstantBuffer](#))

[ID3D12ShaderReflectionType](#)

This shader-reflection interface provides access to variable type. ([ID3D12ShaderReflectionType](#))

[ID3D12ShaderReflectionVariable](#)

This shader-reflection interface provides access to a variable. ([ID3D12ShaderReflectionVariable](#))

[ID3D12SharingContract](#)

Part of a contract between D3D11On12 diagnostic layers and graphics diagnostics tools.

[ID3D12StateObject](#)

Represents a variable amount of configuration state, including shaders, that an application manages as a single unit and which is given to a driver atomically to process, such as compile or optimize.

[ID3D12StateObjectProperties](#)

Provides methods for getting and setting the properties of an ID3D12StateObject.

[ID3D12SwapChainAssistant](#)

[ID3D12Tools](#)

This interface is used to configure the runtime for tools such as PIX. Its not intended or supported for any other scenario.

[ID3D12VersionedRootSignatureDeserializer](#)

Contains methods to return the deserialized D3D12_ROOT_SIGNATURE_DESC1 data structure, of any version of a serialized root signature.

[ID3D12VirtualizationGuestDevice](#)

TBD

[IHolographicCameraInterop](#)

Extends [HolographicCamera](#) to allow 2D texture resources to be created and used as back buffers for holographic rendering in Direct3D 12.

[IHolographicCameraRenderingParametersInterop](#)

A nano-COM interface that allows COM interop with the [HolographicCameraRenderingParameters](#) class for applications that use Direct3D 12 for holographic rendering.

[IHolographicQuadLayerInterop](#)

A nano-COM interface that allows COM interop with the [HolographicQuadLayer](#) Windows Runtime class for apps that use Direct3D 12 for holographic rendering.

[IHolographicQuadLayerUpdateParametersInterop](#)

A nano-COM interface that allows COM interop with the [HolographicQuadLayerUpdateParameters](#) class for applications that use Direct3D 12 for holographic rendering.

Structures

[Expand table](#)

D3D11_RESOURCE_FLAGS	Used with ID3D11On12Device::CreateWrappedResource to override flags that would be inferred by the resource properties or heap properties, including bind flags, misc flags, and CPU access flags.
D3D12_AUTO_BREADCRUMB_NODE	Represents Device Removed Extended Data (DRED) auto-breadcrumb data as a node in a linked list.
D3D12_AUTO_BREADCRUMB_NODE1	
D3D12_BARRIER_GROUP	Describes a group of barriers of a given type.
D3D12_BARRIER_SUBRESOURCE_RANGE	Allows you to transition logically-adjacent ranges of subresources.
D3D12_BLEND_DESC	Describes the blend state. (D3D12_BLEND_DESC)
D3D12_BOX	Describes a 3D box.
D3D12_BUFFER_BARRIER	Describes a buffer memory access barrier. Used by buffer barriers to indicate when resource memory must be made visible for a specific access type.
D3D12_BUFFER_RTV	Describes the elements in a buffer resource to use in a render-target view.
D3D12_BUFFER_SRV	Describes the elements in a buffer resource to use in a shader-resource view.
D3D12_BUFFER_UAV	

Describes the elements in a buffer to use in a unordered-access view. (D3D12_BUFFER_UAV)

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC](#)

Describes a raytracing acceleration structure. Pass this structure into ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure to describe the acceleration structure to be built.

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#)

Defines the inputs for a raytracing acceleration structure build operation. This structure is used by ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure and ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo.

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER](#)

Describes the GPU memory layout of an acceleration structure visualization.

[D3D12_CACHED_PIPELINE_STATE](#)

Stores a pipeline state.

[D3D12_CLEAR_VALUE](#)

Describes a value used to optimize clear operations for a particular resource.

[D3D12_COMMAND_QUEUE_DESC](#)

Describes a command queue.

[D3D12_COMMAND_SIGNATURE_DESC](#)

Describes the arguments (parameters) of a command signature.

[D3D12_COMPUTE_PIPELINE_STATE_DESC](#)

Describes a compute pipeline state object.

[D3D12_CONSTANT_BUFFER_VIEW_DESC](#)

Describes a constant buffer to view.

[D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes a CPU descriptor handle.

[D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS](#)

Describes per-command-list settings used by GPU-Based Validation.

[D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#)

Describes settings used by GPU-Based Validation.

[D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR](#)

Describes the amount of artificial slowdown inserted by the debug device to simulate lower-performance graphics adapters.

[D3D12_DEPTH_STENCIL_DESC](#)

Describes depth-stencil state. (D3D12_DEPTH_STENCIL_DESC)

[D3D12_DEPTH_STENCIL_DESC1](#)

Describes depth-stencil state. (D3D12_DEPTH_STENCIL_DESC1)

[D3D12_DEPTH_STENCIL_VALUE](#)

Specifies a depth and stencil value.

[D3D12_DEPTH_STENCIL_VIEW_DESC](#)

Describes the subresources of a texture that are accessible from a depth-stencil view.

[D3D12_DEPTH_STENCILOP_DESC](#)

Describes stencil operations that can be performed based on the results of stencil test.

[D3D12_DESCRIPTOR_HEAP_DESC](#)

Describes the descriptor heap.

[D3D12_DESCRIPTOR_RANGE](#)

Describes a descriptor range.

[D3D12_DESCRIPTOR_RANGE1](#)

Describes a descriptor range, with flags to determine their volatility.

[D3D12_DEVICE_REMOVED_EXTENDED_DATA](#)

Represents Device Removed Extended Data (DRED) version 1.0 data.

[D3D12_DEVICE_REMOVED_EXTENDED_DATA1](#)

Represents Device Removed Extended Data (DRED) version 1.1 data.

[D3D12_DEVICE_REMOVED_EXTENDED_DATA2](#)

[D3D12_DEVICE_REMOVED_EXTENDED_DATA3](#)

[D3D12_DISCARD_REGION](#)

Describes details for the discard-resource operation.

[D3D12_DISPATCH_ARGUMENTS](#)

Describes dispatch parameters, for use by the compute shader.

[D3D12_DISPATCH_MESH_ARGUMENTS](#)

[D3D12_DISPATCH_RAYS_DESC](#)

Describes the properties of a ray dispatch operation initiated with a call to ID3D12GraphicsCommandList4::DispatchRays.

[D3D12_DRAW_ARGUMENTS](#)

Describes parameters for drawing instances.

[D3D12_DRAW_INDEXED_ARGUMENTS](#)

Describes parameters for drawing indexed instances.

[D3D12_DRED_ALLOCATION_NODE](#)

Describes, as a node in a linked list, data about an allocation tracked by Device Removed Extended Data (DRED).

[D3D12_DRED_ALLOCATION_NODE1](#)

[D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT](#)

Contains a pointer to the head of a linked list of D3D12_AUTO_BREADCRUMB_NODE objects.

[D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1](#)

[D3D12_DRED_BREADCRUMB_CONTEXT](#)

[D3D12_DRED_PAGE_FAULT_OUTPUT](#)

	Describes allocation data related to a GPU page fault on a given virtual address (VA).
D3D12_DRED_PAGE_FAULT_OUTPUT1	
D3D12_DRED_PAGE_FAULT_OUTPUT2	
D3D12_DXIL_LIBRARY_DESC	Describes a DXIL library state subobject that can be included in a state object.
D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION	This subobject is unsupported in the current release.
D3D12_EXISTING_COLLECTION_DESC	A state subobject describing an existing collection that can be included in a state object.
D3D12_EXPORT_DESC	Describes an export from a state subobject such as a DXIL library or a collection state object.
D3D12_FEATURE_DATA_ARCHITECTURE	Provides detail about the adapter architecture, so that your application can better optimize for certain adapter properties.
D3D12_FEATURE_DATA_ARCHITECTURE1	Provides detail about each adapter's architectural details, so that your application can better optimize for certain adapter properties.
D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY	Details the adapter's support for prioritization of different command queue types.
D3D12_FEATURE_DATA_CROSS_NODE	Indicates the level of support for the sharing of resources between different adapters—for example, multiple GPUs.
D3D12_FEATURE_DATA_D3D12_OPTIONS	Describes Direct3D 12 feature options in the current graphics driver.
D3D12_FEATURE_DATA_D3D12_OPTIONS1	Describes the level of support for HLSL 6.0 wave operations.

[D3D12_FEATURE_DATA_D3D12_OPTIONS10](#)

Indicates whether or not the SUM combiner can be used, and whether or not *SV_ShadingRate* can be set from a mesh shader.

[D3D12_FEATURE_DATA_D3D12_OPTIONS11](#)

Indicates whether or not 64-bit integer atomics on resources in descriptor heaps are supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS12](#)

Indicates whether or not Enhanced Barriers are supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS13](#)

TBD

[D3D12_FEATURE_DATA_D3D12_OPTIONS2](#)

Indicates the level of support that the adapter provides for depth-bounds tests and programmable sample positions.

[D3D12_FEATURE_DATA_D3D12_OPTIONS3](#)

Indicates the level of support that the adapter provides for timestamp queries, format-casting, immediate write, view instancing, and barycentrics.

[D3D12_FEATURE_DATA_D3D12_OPTIONS4](#)

Indicates the level of support for 64KB-aligned MSAA textures, cross-API sharing, and native 16-bit shader operations.

[D3D12_FEATURE_DATA_D3D12_OPTIONS5](#)

Indicates the level of support that the adapter provides for render passes, ray tracing, and shader-resource view tier 3 tiled resources.

[D3D12_FEATURE_DATA_D3D12_OPTIONS6](#)

Indicates the level of support that the adapter provides for variable-rate shading (VRS), and indicates whether or not background processing is supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS7](#)

Indicates the level of support that the adapter provides for mesh and amplification shaders, and for sampler feedback.

[D3D12_FEATURE_DATA_D3D12_OPTIONS8](#)

	Indicates whether or not unaligned block-compressed textures are supported.
D3D12_FEATURE_DATA_D3D12_OPTIONS9	Indicates whether or not support exists for mesh shaders, values of <i>SV_RenderTargetArrayIndex</i> that are 8 or greater, typed resource 64-bit integer atomics, derivative and derivative-dependent texture sample operations, and the level of support for WaveMMA (wave_matrix) operations.
D3D12_FEATURE_DATA_DISPLAYABLE	This feature is currently in preview.
D3D12_FEATURE_DATA_EXISTING_HEAPS	Provides detail about whether the adapter supports creating heaps from existing system memory.
D3D12_FEATURE_DATA_FEATURE_LEVELS	Describes info about the feature levels supported by the current graphics driver.
D3D12_FEATURE_DATA_FORMAT_INFO	Describes a DXGI data format and plane count.
D3D12_FEATURE_DATA_FORMAT_SUPPORT	Describes which resources are supported by the current graphics driver for a given format. (D3D12_FEATURE_DATA_FORMAT_SUPPORT)
D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT	Details the adapter's GPU virtual address space limitations, including maximum address bits per resource and per process.
D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS	Describes the multi-sampling image quality levels for a given format and sample count.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT	Indicates the level of support for protected resource sessions.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT	Indicates a count of protected resource session types.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES	Indicates a list of protected resource session types.

[D3D12_FEATURE_DATA_QUERY_META_COMMAND](#)

Indicates the level of support that the adapter provides for metacommands.

[D3D12_FEATURE_DATA_ROOT_SIGNATURE](#)

Indicates root signature version support.

[D3D12_FEATURE_DATA_SERIALIZATION](#)

Indicates the level of support for heap serialization.

[D3D12_FEATURE_DATA_SHADER_CACHE](#)

Describes the level of shader caching supported in the current graphics driver.
(D3D12_FEATURE_DATA_SHADER_CACHE)

[D3D12_FEATURE_DATA_SHADER_MODEL](#)

Contains the supported shader model.

[D3D12_FUNCTION_DESC](#)

Describes a function. (D3D12_FUNCTION_DESC)

[D3D12_GLOBAL_BARRIER](#)

Describes a resource memory access barrier. Used by global, texture, and buffer barriers to indicate when resource memory must be made visible for a specific access type.

[D3D12_GLOBAL_ROOT_SIGNATURE](#)

Defines a global root signature state subobject that will be used with associated shaders.

[D3D12_GPU_DESCRIPTOR_HANDLE](#)

Describes a GPU descriptor handle.

[D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE](#)

Represents a GPU virtual address and indexing stride.

[D3D12_GPU_VIRTUAL_ADDRESS_RANGE](#)

Represents a GPU virtual address range.

[D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE](#)

Represents a GPU virtual address range and stride.

[D3D12_GRAPHICS_PIPELINE_STATE_DESC](#)

Describes a graphics pipeline state object.

[D3D12_HEAP_DESC](#)

Describes a heap.

[D3D12_HEAP_PROPERTIES](#)

Describes heap properties.

[D3D12_HIT_GROUP_DESC](#)

Describes a raytracing hit group state subobject that can be included in a state object.

[D3D12_INDEX_BUFFER_VIEW](#)

Describes the index buffer to view.

[D3D12_INDIRECT_ARGUMENT_DESC](#)

Describes an indirect argument (an indirect parameter), for use with a command signature.

[D3D12_INFO_QUEUE_FILTER](#)

Debug message filter; contains a lists of message types to allow or deny.
(D3D12_INFO_QUEUE_FILTER)

[D3D12_INFO_QUEUE_FILTER_DESC](#)

Allow or deny certain types of messages to pass through a filter.
(D3D12_INFO_QUEUE_FILTER_DESC)

[D3D12_INPUT_ELEMENT_DESC](#)

Describes a single element for the input-assembler stage of the graphics pipeline.

[D3D12_INPUT_LAYOUT_DESC](#)

Describes the input-buffer data for the input-assembler stage.

[D3D12_LIBRARY_DESC](#)

Describes a library. (D3D12_LIBRARY_DESC)

[D3D12_LOCAL_ROOT_SIGNATURE](#)

Defines a local root signature state subobject that will be used with associated shaders.

[D3D12_MEMCPY_DEST](#)

Describes the destination of a memory copy operation.

[D3D12_MESSAGE](#)

A debug message in the Information Queue. (D3D12_MESSAGE)

[D3D12_META_COMMAND_DESC](#)

Describes a meta command.

[D3D12_META_COMMAND_PARAMETER_DESC](#)

Describes a parameter to a meta command.

[D3D12_MIP_REGION](#)

Describes the dimensions of a mip region.

[D3D12_NODE_MASK](#)

A state subobject that identifies the GPU nodes to which the state object applies.

[D3D12_PACKED_MIP_INFO](#)

Describes the tile structure of a tiled resource with mipmaps. (D3D12_PACKED_MIP_INFO)

[D3D12_PARAMETER_DESC](#)

Describes a function parameter. (D3D12_PARAMETER_DESC)

[D3D12_PIPELINE_STATE_STREAM_DESC](#)

Describes a pipeline state stream.

[D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#)

Describes the footprint of a placed subresource, including the offset and the D3D12_SUBRESOURCE_FOOTPRINT.

[D3D12_PROTECTED_RESOURCE_SESSION_DESC](#)

Describes flags for a protected resource session, per adapter.

[D3D12_PROTECTED_RESOURCE_SESSION_DESC1](#)

Describes flags and protection type for a protected resource session, per adapter.

[D3D12_QUERY_DATA_PIPELINE_STATISTICS](#)

Query information about graphics-pipeline activity in between calls to BeginQuery and EndQuery.

[D3D12_QUERY_DATA_PIPELINE_STATISTICS1](#)

[D3D12_QUERY_DATA_SO_STATISTICS](#)

Describes query data for stream output.

[D3D12_QUERY_HEAP_DESC](#)

Describes the purpose of a query heap. A query heap contains an array of individual queries.

[D3D12_RANGE](#)

Describes a memory range.

[D3D12_RANGE_UINT64](#)

Describes a memory range in a 64-bit address space.

[D3D12_RASTERIZER_DESC](#)

Describes rasterizer state. (D3D12_RASTERIZER_DESC)

[D3D12_RAYTRACING_AABB](#)

Represents an axis-aligned bounding box (AABB) used as raytracing geometry.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC](#)

Describes the space requirement for acceleration structure after compaction.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC](#)

Describes the space currently used by an acceleration structure..

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC](#)

Description of the post-build information to generate from an acceleration structure. Use this structure in calls to `EmitRaytracingAccelerationStructurePostbuildInfo` and `BuildRaytracingAccelerationStructure`.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC](#)

Describes the size and layout of the serialized acceleration structure and header.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC](#)

Describes the space requirement for decoding an acceleration structure into a form that can be visualized by tools.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO](#)

Represents prebuild information about a raytracing acceleration structure. Get an instance of this structure by calling `GetRaytracingAccelerationStructurePrebuildInfo`.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV](#)

A shader resource view (SRV) structure for storing a raytracing acceleration structure.

[D3D12_RAYTRACING_GEOMETRY_AABBS_DESC](#)

Describes a set of Axis-aligned bounding boxes that are used in the `D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS` structure to provide input data to a raytracing acceleration structure build operation.

[D3D12_RAYTRACING_GEOMETRY_DESC](#)

Describes a set of geometry that is used in the `D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS` structure to provide input data to a raytracing acceleration structure build operation.

[D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC](#)

Describes a set of triangles used as raytracing geometry. The geometry pointed to by this struct are always in triangle list form, indexed or non-indexed. Triangle strips are not supported.

[D3D12_RAYTRACING_INSTANCE_DESC](#)

Describes an instance of a raytracing acceleration structure used in GPU memory during the acceleration structure build process.

[D3D12_RAYTRACING_PIPELINE_CONFIG](#)

A state subobject that represents a raytracing pipeline configuration.

[D3D12_RAYTRACING_PIPELINE_CONFIG1](#)

A state subobject that represents a raytracing pipeline configuration, with flags.

[D3D12_RAYTRACING_SHADER_CONFIG](#)

A state subobject that represents a shader configuration.

[D3D12_RENDER_PASS_BEGINNING_ACCESS](#)

Describes the access to resource(s) that is requested by an application at the transition into a render pass.

[D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS](#)

Describes the clear value to which resource(s) should be cleared at the beginning of a render pass.

[D3D12_RENDER_PASS_DEPTH_STENCIL_DESC](#)

Describes a binding (fixed for the duration of the render pass) to a depth stencil view (DSV), as well as its beginning and ending access characteristics.

[D3D12_RENDER_PASS_ENDING_ACCESS](#)

Describes the access to resource(s) that is requested by an application at the transition out of a render pass.

[D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS](#)

Describes a resource to resolve to at the conclusion of a render pass.

[D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS](#)

Describes the subresources involved in resolving at the conclusion of a render pass.

[D3D12_RENDER_PASS_RENDER_TARGET_DESC](#)

Describes bindings (fixed for the duration of the render pass) to one or more render target views (RTVs), as well as their beginning and ending access characteristics.

[D3D12_RENDER_TARGET_BLEND_DESC](#)

Describes the blend state for a render target. (D3D12_RENDER_TARGET_BLEND_DESC)

[D3D12_RENDER_TARGET_VIEW_DESC](#)

Describes the subresources from a resource that are accessible by using a render-target view.

[D3D12_RESOURCE_ALIASING_BARRIER](#)

Describes the transition between usages of two different resources that have mappings into the same heap.

[D3D12_RESOURCE_ALLOCATION_INFO](#)

Describes parameters needed to allocate resources.

[D3D12_RESOURCE_ALLOCATION_INFO1](#)

Describes parameters needed to allocate resources, including offset.

[D3D12_RESOURCE_BARRIER](#)

Describes a resource barrier (transition in resource use).

[D3D12_RESOURCE_DESC](#)

Describes a resource, such as a texture. This structure is used extensively.

[D3D12_RESOURCE_DESC1](#)

Describes a resource, such as a texture, including a mip region. This structure is used in several methods.

[D3D12_RESOURCE_TRANSITION_BARRIER](#)

Describes the transition of subresources between different usages.

[D3D12_RESOURCE_UAV_BARRIER](#)

Represents a resource in which all UAV accesses must complete before any future UAV accesses can begin.

[D3D12_ROOT_CONSTANTS](#)

Describes constants inline in the root signature that appear in shaders as one constant buffer.

[D3D12_ROOT_DESCRIPTOR](#)

Describes descriptors inline in the root signature version 1.0 that appear in shaders.

[D3D12_ROOT_DESCRIPTOR_TABLE](#)

Describes the root signature 1.0 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

[D3D12_ROOT_DESCRIPTOR_TABLE1](#)

Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

[D3D12_ROOT_DESCRIPTOR1](#)

Describes descriptors inline in the root signature version 1.1 that appear in shaders.

[D3D12_ROOT_PARAMETER](#)

Describes the slot of a root signature version 1.0.

[D3D12_ROOT_PARAMETER1](#)

Describes the slot of a root signature version 1.1.

[D3D12_ROOT_SIGNATURE_DESC](#)

Describes the layout of a root signature version 1.0.

[D3D12_ROOT_SIGNATURE_DESC1](#)

Describes the layout of a root signature version 1.1.

[D3D12_RT_FORMAT_ARRAY](#)

Wraps an array of render target formats.

[D3D12_SAMPLE_POSITION](#)

Describes a sub-pixel sample position for use with programmable sample positions.

[D3D12_SAMPLER_DESC](#)

Describes a sampler state. (D3D12_SAMPLER_DESC)

[D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER](#)

Opaque data structure describing driver versioning for a serialized acceleration structure.

[D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER](#)

Defines the header for a serialized raytracing acceleration structure.

[D3D12_SHADER_BUFFER_DESC](#)

Describes a shader constant-buffer. (D3D12_SHADER_BUFFER_DESC)

[D3D12_SHADER_BYTCODE](#)

Describes shader data. (D3D12_SHADER_BYTCODE)

[D3D12_SHADER_CACHE_SESSION_DESC](#)

Describes a shader cache session.

[D3D12_SHADER_DESC](#)

Describes a shader. (D3D12_SHADER_DESC)
D3D12_SHADER_INPUT_BIND_DESC
Describes how a shader resource is bound to a shader input. (D3D12_SHADER_INPUT_BIND_DESC)
D3D12_SHADER_RESOURCE_VIEW_DESC
Describes a shader-resource view. (D3D12_SHADER_RESOURCE_VIEW_DESC)
D3D12_SHADER_TYPE_DESC
Describes a shader-variable type. (D3D12_SHADER_TYPE_DESC)
D3D12_SHADER_VARIABLE_DESC
Describes a shader variable. (D3D12_SHADER_VARIABLE_DESC)
D3D12_SIGNATURE_PARAMETER_DESC
Describes a shader signature. (D3D12_SIGNATURE_PARAMETER_DESC)
D3D12_SO_DECLARATION_ENTRY
Describes a vertex element in a vertex buffer in an output slot.
D3D12_STATE_OBJECT_CONFIG
Defines general properties of a state object.
D3D12_STATE_OBJECT_DESC
Description of a state object. Pass this structure into ID3D12Device::CreateStateObject.
D3D12_STATE_SUBOBJECT
Represents a subobject within a state object description. Use with D3D12_STATE_OBJECT_DESC .
D3D12_STATIC_SAMPLER_DESC
Describes a static sampler.
D3D12_STREAM_OUTPUT_BUFFER_VIEW
Describes a stream output buffer.
D3D12_STREAM_OUTPUT_DESC
Describes a streaming output buffer.

[D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION](#)

Associates a subobject defined directly in a state object with shader exports.

[D3D12_SUBRESOURCE_DATA](#)

Describes subresource data. (D3D12_SUBRESOURCE_DATA)

[D3D12_SUBRESOURCE_FOOTPRINT](#)

Describes the format, width, height, depth, and row-pitch of the subresource into the parent resource.

[D3D12_SUBRESOURCE_INFO](#)

Describes subresource data. (D3D12_SUBRESOURCE_INFO)

[D3D12_SUBRESOURCE_RANGE_UINT64](#)

Describes a subresource memory range.

[D3D12_SUBRESOURCE_TILING](#)

Describes a tiled subresource volume. (D3D12_SUBRESOURCE_TILING)

[D3D12_TEX1D_ARRAY_DSV](#)

Describes the subresources from an array of 1D textures to use in a depth-stencil view.

[D3D12_TEX1D_ARRAY_RTV](#)

Describes the subresources from an array of 1D textures to use in a render-target view.

[D3D12_TEX1D_ARRAY_SRV](#)

Describes the subresources from an array of 1D textures to use in a shader-resource view.

[D3D12_TEX1D_ARRAY_UAV](#)

Describes an array of unordered-access 1D texture resources. (D3D12_TEX1D_ARRAY_UAV)

[D3D12_TEX1D_DSV](#)

Describes the subresource from a 1D texture that is accessible to a depth-stencil view.

[D3D12_TEX1D_RTV](#)

Describes the subresource from a 1D texture to use in a render-target view.

[D3D12_TEX1D_SRV](#)

Specifies the subresource from a 1D texture to use in a shader-resource view. (D3D12_TEX1D_SRV)

[D3D12_TEX1D_UAV](#)

Describes a unordered-access 1D texture resource. (D3D12_TEX1D_UAV)

[D3D12_TEX2D_ARRAY_DSV](#)

Describes the subresources from an array of 2D textures that are accessible to a depth-stencil view.

[D3D12_TEX2D_ARRAY_RTV](#)

Describes the subresources from an array of 2D textures to use in a render-target view.
(D3D12_TEX2D_ARRAY_RTV)

[D3D12_TEX2D_ARRAY_SRV](#)

Describes the subresources from an array of 2D textures to use in a shader-resource view.
(D3D12_TEX2D_ARRAY_SRV)

[D3D12_TEX2D_ARRAY_UAV](#)

Describes an array of unordered-access 2D texture resources. (D3D12_TEX2D_ARRAY_UAV)

[D3D12_TEX2D_DSV](#)

Describes the subresource from a 2D texture that is accessible to a depth-stencil view.

[D3D12_TEX2D_RTV](#)

Describes the subresource from a 2D texture to use in a render-target view. (D3D12_TEX2D_RTV)

[D3D12_TEX2D_SRV](#)

Describes the subresource from a 2D texture to use in a shader-resource view. (D3D12_TEX2D_SRV)

[D3D12_TEX2D_UAV](#)

Describes a unordered-access 2D texture resource. (D3D12_TEX2D_UAV)

[D3D12_TEX2DMS_ARRAY_DSV](#)

Describes the subresources from an array of multi sampled 2D textures for a depth-stencil view.

[D3D12_TEX2DMS_ARRAY_RTV](#)

Describes the subresources from an array of multi sampled 2D textures to use in a render-target view.

[D3D12_TEX2DMS_ARRAY_SRV](#)

Describes the subresources from an array of multi sampled 2D textures to use in a shader-resource view.

[D3D12_TEX2DMS_DSV](#)

Describes the subresource from a multi sampled 2D texture that is accessible to a depth-stencil view.

[D3D12_TEX2DMS_RTV](#)

Describes the subresource from a multi sampled 2D texture to use in a render-target view.

[D3D12_TEX2DMS_SRV](#)

Describes the subresources from a multi sampled 2D texture to use in a shader-resource view.

[D3D12_TEX3D_RTV](#)

Describes the subresources from a 3D texture to use in a render-target view.

[D3D12_TEX3D_SRV](#)

Describes the subresources from a 3D texture to use in a shader-resource view.

[D3D12_TEX3D_UAV](#)

Describes a unordered-access 3D texture resource. (D3D12_TEX3D_UAV)

[D3D12_TEXCUBE_ARRAY_SRV](#)

Describes the subresources from an array of cube textures to use in a shader-resource view.

[D3D12_TEXCUBE_SRV](#)

Describes the subresource from a cube texture to use in a shader-resource view.

[D3D12_TEXTURE_BARRIER](#)

Expresses an access transition for a texture.

[D3D12_TEXTURE_COPY_LOCATION](#)

Describes a portion of a texture for the purpose of texture copies.

D3D12_TILE_REGION_SIZE	Describes the size of a tiled region. (D3D12_TILE_REGION_SIZE)
D3D12_TILE_SHAPE	Describes the shape of a tile by specifying its dimensions. (D3D12_TILE_SHAPE)
D3D12_TILED_RESOURCE_COORDINATE	Describes the coordinates of a tiled resource. (D3D12_TILED_RESOURCE_COORDINATE)
D3D12_UNORDERED_ACCESS_VIEW_DESC	Describes the subresources from a resource that are accessible by using an unordered-access view.
D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA	Represents versioned Device Removed Extended Data (DRED) data.
D3D12_VERSIONED_ROOT_SIGNATURE_DESC	Holds any version of a root signature description, and is designed to be used with serialization/deserialization functions.
D3D12_VERTEX_BUFFER_VIEW	Describes a vertex buffer view.
D3D12_VIEW_INSTANCE_LOCATION	Specifies the viewport/stencil and render target associated with a view instance.
D3D12_VIEW_INSTANCING_DESC	Specifies parameters used during view instancing configuration.
D3D12_VIEWPORT	Describes the dimensions of a viewport.
D3D12_WRITEBUFFERIMMEDIATE_PARAMETER	Specifies the immediate value and destination address written using ID3D12CommandList2::WriteBufferImmediate.

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

d3d11on12.h header

Article 01/24/2023

This header is used by Direct3D 12 Graphics. For more information, see:

- [Direct3D 12 Graphics](#)

d3d11on12.h contains the following programming interfaces:

Interfaces

[ID3D11On12Device](#)

Handles the creation, wrapping, and releasing of D3D11 resources for Direct3D11on12.

[ID3D11On12Device1](#)

Enables better interoperability with a component that might be handed a Direct3D 11 device, but which wants to leverage Direct3D 12 instead.

[ID3D11On12Device2](#)

Enables you to take resources created through the Direct3D 11 APIs, and use them in Direct3D 12.

Functions

[D3D11On12CreateDevice](#)

Creates a device that uses Direct3D 11 functionality in Direct3D 12, specifying a pre-existing Direct3D 12 device to use for Direct3D 11 interop.

Structures

[D3D11_RESOURCE_FLAGS](#)

Used with ID3D11On12Device::CreateWrappedResource to override flags that would be inferred by the resource properties or heap properties, including bind flags, misc flags, and CPU access flags.

Feedback

Was this page helpful?

 Yes

 No

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

D3D11_RESOURCE_FLAGS structure (d3d11on12.h)

Article 10/05/2021

Used with [ID3D11On12Device::CreateWrappedResource](#) to override flags that would be inferred by the resource properties or heap properties, including bind flags, misc flags, and CPU access flags.

Syntax

C++

```
typedef struct D3D11_RESOURCE_FLAGS {
    UINT BindFlags;
    UINT MiscFlags;
    UINT CPUAccessFlags;
    UINT StructureByteStride;
} D3D11_RESOURCE_FLAGS;
```

Members

BindFlags

Bind flags must be either completely inferred, or completely specified, to allow the graphics driver to scope a general D3D12 resource to something that D3D11 can understand.

If a bind flag is specified which is not supported by the provided resource, an error will be returned.

The following bind flags ([D3D11_BIND_FLAG](#) enumeration constants) will not be assumed, and must be specified in order for a resource to be used in such a fashion:

- D3D11_BIND_VERTEX_BUFFER
- D3D11_BIND_INDEX_BUFFER
- D3D11_BIND_CONSTANT_BUFFER
- D3D11_BIND_STREAM_OUTPUT
- D3D11_BIND_DECODER
- D3D11_BIND_VIDEO_ENCODER

The following bind flags will be assumed based on the presence of the corresponding D3D12 resource flag, and can be removed by specifying bind flags:

- D3D11_BIND_SHADER_RESOURCE, as long as D3D12_RESOURCE_MISC_DENY_SHADER_RESOURCE is not present
- D3D11_BIND_RENDER_TARGET, if D3D12_RESOURCE_MISC_ALLOW_RENDER_TARGET is present
- D3D11_BIND_DEPTH_STENCIL, if D3D12_RESOURCE_MISC_ALLOW_DEPTH_STENCIL is present
- D3D11_BIND_UNORDERED_ACCESS, if D3D12_RESOURCE_MISC_ALLOW_UNORDERED_ACCESS is present

A render target or UAV buffer can be wrapped without overriding flags; but a VB/IB/CB/SO buffer must have bind flags manually specified, since these are mutually exclusive in Direct3D 11.

MiscFlags

If misc flags are nonzero, then any specified flags will be OR'd into the final resource desc with inferred flags. Misc flags can be partially specified in order to add functionality, but misc flags which are implied cannot be masked out.

The following misc flags ([D3D11_RESOURCE_MISC_FLAG](#) enumeration constants) will not be assumed:

- D3D11_RESOURCE_MISC_GENERATE_MIPS (conflicts with CLAMP).
- D3D11_RESOURCE_MISC_TEXTURECUBE (alters default view behavior).
- D3D11_RESOURCE_MISC_DRAWINDIRECT_ARGS (exclusive with some bind flags).
- D3D11_RESOURCE_MISC_BUFFER_ALLOW_RAW_VIEWS (exclusive with other types of UAVs).
- D3D11_RESOURCE_MISC_BUFFER_STRUCTURED (exclusive with other types of UAVs).
- D3D11_RESOURCE_MISC_RESOURCE_CLAMP (prohibits D3D10 QIs, conflicts with GENERATE_MIPS).
- D3D11_RESOURCE_MISC_SHARED_KEYEDMUTEX. It is possible to create a D3D11 keyed mutex resource, create a shared handle for it, and open it via 11on12 or D3D11.

The following misc flags will be assumed, and cannot be removed from the produced resource desc. If one of these is set, and the D3D12 resource does not support it, creation will fail:

- D3D11_RESOURCE_MISC_SHARED, D3D11_RESOURCE_MISC_SHARED_NTHANDLE, D3D11_RESOURCE_MISC_RESTRICT_SHARED_RESOURCE, if appropriate heap misc flags are present.
- D3D11_RESOURCE_MISC_GDI_COMPATIBLE, if D3D12 resource is GDI-compatible.
- D3D11_RESOURCE_MISC_TILED, if D3D12 resource was created via [CreateReservedResource](#).
- D3D11_RESOURCE_MISC_TILE_POOL, if a D3D12 heap was passed in.

The following misc flags are invalid to specify for this API:

- D3D11_RESOURCE_MISC_RESTRICTED_CONTENT, since D3D12 only supports hardware protection.
- D3D11_RESOURCE_MISC_RESTRICT_SHARED_RESOURCE_DRIVER does not exist in 12, and cannot be added in after resource creation.
- D3D11_RESOURCE_MISC_GUARDED is only meant to be set by an internal creation mechanism.

CPUAccessFlags

The **CPUAccessFlags** are not inferred from the D3D12 resource. This is because all resources are treated as D3D11_USAGE_DEFAULT, so **CPUAccessFlags** force validation which assumes [Map](#) of default buffers or textures. Wrapped resources do not support [Map\(DISCARD\)](#). Wrapped resources do not support [Map\(NO_OVERWRITE\)](#), but that can be implemented by mapping the underlying D3D12 resource instead. Issuing a [Map](#) call on a wrapped resource will synchronize with all D3D11 work submitted against that resource, unless the DO_NOT_WAIT flag was used.

StructureByteStride

The size of each element in the buffer structure (in bytes) when the buffer represents a structured buffer.

Remarks

Use this structure with [CreateWrappedResource](#).

Requirements

[\[\]](#) Expand table

Requirement	Value
Header	d3d11on12.h

See also

[11on12 Structures](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D11On12CreateDevice function (d3d11on12.h)

Article 05/27/2022

Creates a device that uses Direct3D 11 functionality in Direct3D 12, specifying a pre-existing Direct3D 12 device to use for Direct3D 11 interop.

Syntax

C++

```
HRESULT D3D11On12CreateDevice(
    [in]           IUnknown*          *pDevice,
                           UINT            Flags,
    [in, optional] const D3D_FEATURE_LEVEL* pFeatureLevels,
                           UINT            FeatureLevels,
    [in, optional] IUnknown*          *ppCommandQueues,
                           UINT            NumQueues,
                           UINT            NodeMask,
    [out, optional] ID3D11Device*      **ppDevice,
    [out, optional] ID3D11DeviceContext* **ppImmediateContext,
    [out, optional] D3D_FEATURE_LEVEL*   *pChosenFeatureLevel
);
```

Parameters

`[in] pDevice`

Type: `IUnknown*`

Specifies a pre-existing Direct3D 12 device to use for Direct3D 11 interop. May not be NULL.

`Flags`

Type: `UINT`

One or more bitwise OR'd flags from `D3D11_CREATE_DEVICE_FLAG`. These are the same flags as those used by `D3D11CreateDeviceAndSwapChain`. Specifies which runtime layers to enable. `Flags` must be compatible with device flags, and its `NodeMask` must be a subset of the `NodeMask` provided to the present API.

`[in, optional] pFeatureLevels`

Type: **const D3D_FEATURE_LEVEL***

An array of any of the following:

- D3D_FEATURE_LEVEL_12_1
- D3D_FEATURE_LEVEL_12_0
- D3D_FEATURE_LEVEL_11_1
- D3D_FEATURE_LEVEL_11_0
- D3D_FEATURE_LEVEL_10_1
- D3D_FEATURE_LEVEL_10_0
- D3D_FEATURE_LEVEL_9_3
- D3D_FEATURE_LEVEL_9_2
- D3D_FEATURE_LEVEL_9_1

The first feature level that is less than or equal to the Direct3D 12 device's feature level will be used to perform Direct3D 11 validation. Creation will fail if no acceptable feature levels are provided. Providing NULL will default to the Direct3D 12 device's feature level.

FeatureLevels

Type: **UINT**

The size of (that is, the number of elements in) the *pFeatureLevels* array.

[in, optional] ppCommandQueues

Type: **IUnknown* const ***

An array of unique queues for D3D11On12 to use. The queues must be of the 3D command queue type.

NumQueues

Type: **UINT**

The size of (that is, the number of elements in) the *ppCommandQueues* array.

NodeMask

Type: **UINT**

Which node of the Direct3D 12 device to use. Only 1 bit may be set.

[out, optional] ppDevice

Type: **ID3D11Device****

Pointer to the returned [ID3D11Device](#). May be NULL.

[out, optional] ppImmediateContext

Type: [ID3D11DeviceContext**](#)

A pointer to the returned [ID3D11DeviceContext](#). May be NULL.

[out, optional] pChosenFeatureLevel

Type: [D3D_FEATURE_LEVEL*](#)

A pointer to the returned feature level. May be NULL.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#) that are documented for [D3D11CreateDevice](#).

This method returns [DXGI_ERROR_SDK_COMPONENT_MISSING](#) if you specify [D3D11_CREATE_DEVICE_DEBUG](#) in *Flags* and the incorrect version of the [debug layer](#) is installed on your computer. Install the latest Windows SDK to get the correct version.

Remarks

The function signature [PFN_D3D11ON12_CREATE_DEVICE](#) is provided as a typedef, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

Examples

To render text over Direct3D 12 using Direct2D via the 11On12 device, load the rendering pipeline dependencies.

C++

```
// Load the rendering pipeline dependencies.
void D3D1211on12::LoadPipeline()
{
    UINT d3d11DeviceFlags = D3D11_CREATE_DEVICE_BGRA_SUPPORT;
    D2D1_FACTORY_OPTIONS d2dFactoryOptions = {};
#if defined(_DEBUG)
    // Enable the D2D debug layer.
```

```

d2dFactoryOptions.debugLevel = D2D1_DEBUG_LEVEL_INFORMATION;

// Enable the Direct3D 11 debug layer.
d3d11DeviceFlags |= D3D11_CREATE_DEVICE_DEBUG;

// Enable the Direct3D 12 debug layer.
{
    ComPtr<ID3D12Debug> debugController;
    if
(SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController)))
    {
        debugController->EnableDebugLayer();
    }
}
#endif

ComPtr<IDXGIFactory4> factory;
ThrowIfFailed(CreateDXGIFactory1(IID_PPV_ARGS(&factory)));

if (m_useWarpDevice)
{
    ComPtr<IDXGIAdapter> warpAdapter;
    ThrowIfFailed(factory->EnumWarpAdapter(IID_PPV_ARGS(&warpAdapter)));

    ThrowIfFailed(D3D12CreateDevice(
        warpAdapter.Get(),
        D3D_FEATURE_LEVEL_11_0,
        IID_PPV_ARGS(&m_d3d12Device)
    ));
}
else
{
    ComPtr<IDXGIAdapter1> hardwareAdapter;
    GetHardwareAdapter(factory.Get(), &hardwareAdapter);

    ThrowIfFailed(D3D12CreateDevice(
        hardwareAdapter.Get(),
        D3D_FEATURE_LEVEL_11_0,
        IID_PPV_ARGS(&m_d3d12Device)
    ));
}

// Describe and create the command queue.
D3D12_COMMAND_QUEUE_DESC queueDesc = {};
queueDesc.Flags = D3D12_COMMAND_QUEUE_FLAG_NONE;
queueDesc.Type = D3D12_COMMAND_LIST_TYPE_DIRECT;

ThrowIfFailed(m_d3d12Device->CreateCommandQueue(&queueDesc,
IID_PPV_ARGS(&m_commandQueue)));

// Describe the swap chain.
DXGI_SWAP_CHAIN_DESC swapChainDesc = {};
swapChainDesc.BufferCount = FrameCount;
swapChainDesc.BufferDesc.Width = m_width;
swapChainDesc.BufferDesc.Height = m_height;

```

```

swapChainDesc.BufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
swapChainDesc.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
swapChainDesc.SwapEffect = DXGI_SWAP_EFFECT_FLIP_DISCARD;
swapChainDesc.OutputWindow = Win32Application::GetHwnd();
swapChainDesc.SampleDesc.Count = 1;
swapChainDesc.Windowed = TRUE;

ComPtr swapChain;
ThrowIfFailed(factory->CreateSwapChain(
    m_commandQueue.Get(),           // Swap chain needs the queue so that
it can force a flush on it.
    &swapChainDesc,
    &swapChain
));

ThrowIfFailed(swapChain.As(&m_swapChain));

// This sample does not support fullscreen transitions.
ThrowIfFailed(factory-
>MakeWindowAssociation(Win32Application::GetHwnd(), DXGI_MWA_NO_ALT_ENTER));

m_frameIndex = m_swapChain->GetCurrentBackBufferIndex();

// Create an 11 device wrapped around the 12 device and share
// 12's command queue.
ComPtr d3d11Device;
ThrowIfFailed(D3D11On12CreateDevice(
    m_d3d12Device.Get(),
    D3D11DeviceFlags,
    nullptr,
    0,
    reinterpret_cast<IUnknown**>(m_commandQueue.GetAddressOf()),
    1,
    0,
    &d3d11Device,
    &m_d3d11DeviceContext,
    nullptr
));

// Query the 110n12 device from the 11 device.
ThrowIfFailed(d3d11Device.As(&m_d3d11On12Device));

// Create D2D/DWrite components.
{
    D2D1_DEVICE_CONTEXT_OPTIONS deviceOptions =
D2D1_DEVICE_CONTEXT_OPTIONS_NONE;
    ThrowIfFailed(D2D1CreateFactory(
        D2D1_FACTORY_TYPE_SINGLE_THREADED,
        __uuidof(ID2D1Factory3),
        &d2dFactoryOptions,
        &m_d2dFactory));
    ComPtr dxgiDevice;
    ThrowIfFailed(m_d3d11On12Device.As(&dxgiDevice));
    ThrowIfFailed(m_d2dFactory->CreateDevice(dxgiDevice.Get(),
&m_d2dDevice));
}

```

```

        ThrowIfFailed(m_d2dDevice->CreateDeviceContext(deviceOptions,
&m_d2dDeviceContext));
        ThrowIfFailed(DWriteCreateFactory(
            DWRITE_FACTORY_TYPE_SHARED,
            __uuidof(IDWriteFactory),
            &m_dWriteFactory));
    }

    // Query the desktop's dpi settings, which will be used to create
    // D2D's render targets.
    float dpi = GetDpiForWindow(Win32Application::GetHwnd());
    D2D1_BITMAP_PROPERTIES1 bitmapProperties = D2D1::BitmapProperties1(
        D2D1_BITMAP_OPTIONS_TARGET | D2D1_BITMAP_OPTIONS_CANNOT_DRAW,
        D2D1::PixelFormat(DXGI_FORMAT_UNKNOWN,
D2D1_ALPHA_MODE_PREMULTIPLIED),
        dpi,
        dpi
    );

    // Create descriptor heaps.
    {
        // Describe and create a render target view (RTV) descriptor heap.
        D3D12_DESCRIPTOR_HEAP_DESC rtvHeapDesc = {};
        rtvHeapDesc.NumDescriptors = FrameCount;
        rtvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_RTV;
        rtvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
        ThrowIfFailed(m_d3d12Device->CreateDescriptorHeap(&rtvHeapDesc,
IID_PPV_ARGS(&m_rtvHeap)));

        m_rtvDescriptorSize =
            m_d3d12Device-
>GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_RTV);
    }

    // Create frame resources.
    {
        CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart());

        // Create a RTV, D2D render target, and a command allocator for each
frame.
        for (UINT n = 0; n < FrameCount; n++)
        {
            ThrowIfFailed(m_swapChain->GetBuffer(n,
IID_PPV_ARGS(&m_renderTargets[n])));
            m_d3d12Device->CreateRenderTargetView(m_renderTargets[n].Get(),
nullptr, rtvHandle);

            // Create a wrapped 110n12 resource of this back buffer. Since
we are
                // rendering all Direct3D 12 content first and then all D2D
content, we specify
                    // the In resource state as RENDER_TARGET - because Direct3D 12
will have last
                        // used it in this state - and the Out resource state as

```

```

PRESENT. When
    // ReleaseWrappedResources() is called on the 11on12 device, the
resource
    // will be transitioned to the PRESENT state.
D3D11_RESOURCE_FLAGS d3d11Flags = { D3D11_BIND_RENDER_TARGET };
ThrowIfFailed(m_d3d11on12Device->CreateWrappedResource(
    m_renderTargets[n].Get(),
    &d3d11Flags,
    D3D12_RESOURCE_STATE_RENDER_TARGET,
    D3D12_RESOURCE_STATE_PRESENT,
    IID_PPV_ARGS(&m_wrappedBackBuffers[n])
));

// Create a render target for D2D to draw directly to this back
buffer.
ComPtr<IDXGISurface> surface;
ThrowIfFailed(m_wrappedBackBuffers[n].As(&surface));
ThrowIfFailed(m_d2dDeviceContext->CreateBitmapFromDxgiSurface(
    surface.Get(),
    &bitmapProperties,
    &m_d2dRenderTarget[n]
));

rtvHandle.Offset(1, m_rtvDescriptorSize);

ThrowIfFailed(m_d3d12Device->CreateCommandAllocator(
    D3D12_COMMAND_LIST_TYPE_DIRECT,
    IID_PPV_ARGS(&m_commandAllocators[n])));
}
}
}

```

See the notes about the [Example code in the Direct3D 12 reference content](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device interface (d3d11on12.h)

Article 02/22/2024

Handles the creation, wrapping, and releasing of Direct3D 11 resources for Direct3D11on12.

Inheritance

The **ID3D11On12Device** interface inherits from the [IUnknown interface](#).

Inheritance

The **ID3D11On12Device** interface inherits from the [IUnknown interface](#).

Methods

The **ID3D11On12Device** interface has these methods.

[+] Expand table

ID3D11On12Device::AcquireWrappedResources
Acquires D3D11 resources for use with D3D 11on12. Indicates that rendering to the wrapped resources can begin again.
ID3D11On12Device::CreateWrappedResource
This method creates D3D11 resources for use with D3D 11on12.
ID3D11On12Device::ReleaseWrappedResources
Releases D3D11 resources that were wrapped for D3D 11on12.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11on12.h

See also

- [11on12 Interfaces](#)
- [IUnknown interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device::AcquireWrappedResources method (d3d11on12.h)

Article 02/22/2024

Acquires D3D11 resources for use with D3D 11on12. Indicates that rendering to the wrapped resources can begin again.

Syntax

C++

```
void AcquireWrappedResources(
    [in] ID3D11Resource * const *ppResources,
    UINT                 NumResources
);
```

Parameters

[in] ppResources

Type: [ID3D11Resource*](#)

Specifies a pointer to a set of D3D11 resources, defined by [ID3D11Resource](#).

NumResources

Type: [UINT](#)

Count of the number of resources.

Return value

None

Remarks

This method marks the resources as "acquired" in hazard tracking.

Keyed mutex resources cannot be provided to this method; use [IDXGIFactory2::AcquireSync](#) instead.

Examples

Render text over D3D12 using D2D via the 11On12 device.

C++

```
// Render text over D3D12 using D2D via the 11On12 device.
void D3D1211on12::RenderUI()
{
    D2D1_SIZE_F rtSize = m_d2dRenderTargets[m_frameIndex]->GetSize();
    D2D1_RECT_F textRect = D2D1::RectF(0, 0, rtSize.width, rtSize.height);
    static const WCHAR text[] = L"11On12";

    // Acquire our wrapped render target resource for the current back
    // buffer.
    m_d3d110n12Device-
>AcquireWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(),
1);

    // Render text directly to the back buffer.
    m_d2dDeviceContext->SetTarget(m_d2dRenderTargets[m_frameIndex].Get());
    m_d2dDeviceContext->BeginDraw();
    m_d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());
    m_d2dDeviceContext->DrawTextW(
        text,
        _countof(text) - 1,
        m_textFormat.Get(),
        &textRect,
        m_textBrush.Get()
    );
    ThrowIfFailed(m_d2dDeviceContext->EndDraw());

    // Release our wrapped render target resource. Releasing
    // transitions the back buffer resource to the state specified
    // as the OutState when the wrapped resource was created.
    m_d3d110n12Device-
>ReleaseWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(),
1);

    // Flush to submit the 11 command list to the shared command queue.
    m_d3d11DeviceContext->Flush();
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

[ID3D11On12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device::CreateWrappedResource method (d3d11on12.h)

Article 02/22/2024

This method creates D3D11 resources for use with D3D 11on12.

Syntax

C++

```
HRESULT CreateWrappedResource(
    [in]          IUnknown*           *pResource12,
    [in]          const D3D11_RESOURCE_FLAGS* pFlags11,
    D3D12_RESOURCE_STATES        InState,
    D3D12_RESOURCE_STATES        OutState,
    REFIID                      riid,
    [out, optional] void**        ppResource11
);
```

Parameters

[in] pResource12

Type: **IUnknown***

A pointer to an already-created D3D12 resource or heap.

[in] pFlags11

Type: **const D3D11_RESOURCE_FLAGS***

A **D3D11_RESOURCE_FLAGS** structure that enables an application to override flags that would be inferred by the resource/heap properties. The **D3D11_RESOURCE_FLAGS** structure contains bind flags, misc flags, and CPU access flags.

InState

Type: **D3D12_RESOURCE_STATES**

The use of the resource on input, as a bitwise-OR'd combination of **D3D12_RESOURCE_STATES** enumeration constants.

OutState

Type: [D3D12_RESOURCE_STATES](#)

The use of the resource on output, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

`riid`

Type: [REFIID](#)

The globally unique identifier (**GUID**) for the wrapped resource interface. The **REFIID**, or **GUID**, of the interface to the wrapped resource can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D11Resource)` will get the **GUID** of the interface to a wrapped resource.

`[out, optional] ppResource11`

Type: [void**](#)

After the method returns, points to the newly created wrapped D3D11 resource or heap.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

[ID3D11On12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device::ReleaseWrappedResources method (d3d11on12.h)

Article 02/22/2024

Releases D3D11 resources that were wrapped for D3D 11on12.

Syntax

C++

```
void ReleaseWrappedResources(
    [in] ID3D11Resource * const *ppResources,
    UINT                 NumResources
);
```

Parameters

[in] ppResources

Type: [ID3D11Resource*](#)

Specifies a pointer to a set of D3D11 resources, defined by [ID3D11Resource](#).

NumResources

Type: [UINT](#)

Count of the number of resources.

Return value

None

Remarks

Call this method prior to calling Flush, to insert resource barriers to the appropriate "out" state, and to mark that they should then be expected to be in the "in" state. If no resource list is provided, all wrapped resources are transitioned. These resources will be marked as "not acquired" in hazard tracking until [ID3D11On12Device::AcquireWrappedResources](#) is called.

Keyed mutex resources cannot be provided to this method; use [IDXGIFactory::ReleaseSync](#) instead.

Examples

Render text over D3D12 using D2D via the 11On12 device.

C++

```
// Render text over D3D12 using D2D via the 11On12 device.
void D3D1211on12::RenderUI()
{
    D2D1_SIZE_F rtSize = m_d2dRenderTargets[m_frameIndex]->GetSize();
    D2D1_RECT_F textRect = D2D1::RectF(0, 0, rtSize.width, rtSize.height);
    static const WCHAR text[] = L"11On12";

    // Acquire our wrapped render target resource for the current back
    // buffer.
    m_d3d110n12Device-
    >AcquireWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(),
    1);

    // Render text directly to the back buffer.
    m_d2dDeviceContext->SetTarget(m_d2dRenderTargets[m_frameIndex].Get());
    m_d2dDeviceContext->BeginDraw();
    m_d2dDeviceContext->SetTransform(D2D1::Matrix3x2F::Identity());
    m_d2dDeviceContext->DrawTextW(
        text,
        _countof(text) - 1,
        m_textFormat.Get(),
        &textRect,
        m_textBrush.Get()
    );
    ThrowIfFailed(m_d2dDeviceContext->EndDraw());

    // Release our wrapped render target resource. Releasing
    // transitions the back buffer resource to the state specified
    // as the OutState when the wrapped resource was created.
    m_d3d110n12Device-
    >ReleaseWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(),
    1);

    // Flush to submit the 11 command list to the shared command queue.
    m_d3d11DeviceContext->Flush();
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

[ID3D11On12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D11On12Device1 interface (d3d11on12.h)

Article 02/22/2024

Enables better interoperability with a component that might be handed a Direct3D 11 device, but which wants to leverage Direct3D 12 instead. This interface extends [ID3D11On12Device](#) to retrieve the [Direct3D 12 device](#) being interoperated with.

Inheritance

The [ID3D11On12Device1](#) interface inherits from the [ID3D11On12Device](#) interface.

Inheritance

The [ID3D11On12Device1](#) interface inherits from the [ID3D11On12Device](#) interface.

Methods

The [ID3D11On12Device1](#) interface has these methods.

[+] [Expand table](#)

ID3D11On12Device1::GetD3D12Device
Retrieves the Direct3D 12 device being interoperated with.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1903
Minimum supported server	Windows Server, version 1903
Header	d3d11on12.h

See also

- [11on12 Interfaces](#)
 - [ID3D11On12Device interface](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device1::GetD3D12Device method (d3d11on12.h)

Article02/22/2024

Retrieves the [Direct3D 12 device](#) being interoperated with. This enables better interoperability with a component that might be handed a Direct3D 11 device, but which wants to leverage Direct3D 12 instead.

Syntax

C++

```
HRESULT GetD3D12Device(  
    REFIID riid,  
    void    **ppvDevice  
>;
```

Parameters

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppvDevice`. This is expected to be the GUID of [ID3D12Device](#).

`ppvDevice`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the device. This is the address of a pointer to an [ID3D12Device](#), representing the Direct3D 12 device.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

- [ID3D12Device interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device2 interface (d3d11on12.h)

Article02/16/2023

Enables you to take resources created through the Direct3D 11 APIs, and use them in Direct3D 12. This interface extends [ID3D11On12Device1](#).

Inheritance

The [ID3D11On12Device2](#) interface inherits from the [ID3D11On12Device1](#) interface.

Inheritance

The [ID3D11On12Device2](#) interface inherits from the [ID3D11On12Device1](#) interface.

Methods

The [ID3D11On12Device2](#) interface has these methods.

[+] Expand table

ID3D11On12Device2::ReturnUnderlyingResource
With this method, you can return a Direct3D 11 resource object to Direct3D11On12, and indicate when the resource will be ready to consume.
ID3D11On12Device2::UnwrapUnderlyingResource
Unwraps a Direct3D 11 resource object, and retrieves it as a Direct3D 12 resource object.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)

Requirement	Value
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	d3d11on12.h

See also

- [11on12 interfaces](#)
- [ID3D11On12Device1 interface](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device2::ReturnUnderlyingResource method (d3d11on12.h)

Article 02/22/2024

With this method, you can return a Direct3D 11 resource object to Direct3D11On12, and indicate (by way of fences and fence signal values) when the resource will be ready for Direct3D11On12 to consume. You should call **ReturnUnderlyingResource** once Direct3D 12 work has been scheduled.

Syntax

C++

```
HRESULT ReturnUnderlyingResource(
    [in] ID3D11Resource *pResource11,
    [in] UINT           NumSync,
    [in] UINT64         *pSignalValues,
    [in] ID3D12Fence   **ppFences
);
```

Parameters

[in] pResource11

Type: [ID3D11Resource*](#)

The Direct3D 11 resource object that you wish to return.

[in] NumSync

Type: [UINT](#)

The number of elements in the arrays pointed to by *pSignalValues* and *ppFences*.

[in] pSignalValues

Type: [UINT64*](#)

A pointer to an array of fence signal values.

[in] ppFences

Type: [ID3D12Fence**](#)

A pointer to an array of fence objects.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

Remarks

When you return a resource, you provide a set of fences and fence signal values whose completion indicates that the resource is back in the [D3D12_RESOURCE_STATE_COMMON](#) state, and ready for Direct3D11On12 to consume it.

In the parallel arrays *pSignalValues* and *ppFences*, include any pending work against the resource. The Direct3D11On12 translation layer defers the waits for these arguments until work is scheduled against the resource.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

- [ID3D11On12Device2::UnwrapUnderlyingResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D11On12Device2::UnwrapUnderlyingResource method (d3d11on12.h)

Article 10/13/2021

Unwraps a Direct3D 11 resource object, and retrieves it as a Direct3D 12 resource object.

Syntax

C++

```
HRESULT UnwrapUnderlyingResource(
    [in] ID3D11Resource     *pResource11,
    [in] ID3D12CommandQueue *pCommandQueue,
    [in] REFIID              riid,
    [out] void                **ppvResource12
);
```

Parameters

[in] pResource11

Type: [ID3D11Resource*](#)

The Direct3D 11 resource object to unwrap.

[in] pCommandQueue

Type: [ID3D12CommandQueue*](#)

The command queue on which your application plans to use the resource. Any pending work accessing the resource causes fence waits to be scheduled on this queue. You can then queue further work on this queue, including a signal on a caller-owned fence.

[in] riid

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppvResource12`.

[out] ppvResource12

Type: **void****

A pointer to a memory block that receives a pointer to the Direct3D 12 resource.

Return value

Type: **HRESULT**

If the function succeeds, it returns **S_OK**. Otherwise, it returns an [HRESULT error code](#).

Remarks

The resource is transitioned to **D3D12_RESOURCE_STATE_COMMON** (if it wasn't already in that state), and appropriate waits are inserted into the command queue (*pCommandQueue*).

There are some restrictions on what can be unwrapped: no keyed mutex resources, no GDI-compatible resources, and no buffers. However, you can use **UnwrapUnderlyingResource** to unwrap resources created through the [**ID3D11On12Device::CreateWrappedResource**](#) method, as well as resources created through [**ID3D11Device::CreateTexture2D**](#).

In general, you must return the object to Direct3D11on12 before using it again in Direct3D 11 (see [**ID3D11On12Device2::ReturnUnderlyingResource**](#)).

You can also use **UnwrapUnderlyingResource** to unwrap a swapchain buffer. You must also return the resource to Direct3D11on12 before calling **Present** (or otherwise using the resource).

Unwrapping a resource checks out the resource from the Direct3D11On12 translation layer. You may not schedule any translation layer usage (through either version of the API) while the resource is checked out. Check the resource back in (also known as *returning* the resource) with [**ID3D11On12Device2::ReturnUnderlyingResource**](#).

UnwrapUnderlyingResource doesn't flush, and it may schedule GPU work. You should flush after calling **UnwrapUnderlyingResource** if you externally wait for completion.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	d3d11on12.h
Library	D3D11.lib
DLL	D3D11.dll

See also

- [ID3D12Device interface](#)
- [ID3D11On12Device2::ReturnUnderlyingResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

d3d12.h header

Article02/10/2023

This header is used by Direct3D 12 Graphics. For more information, see:

- [Direct3D 12 Graphics](#)

d3d12.h contains the following programming interfaces:

Interfaces

ID3D12CommandAllocator
Represents the allocations of storage for graphics processing unit (GPU) commands.
ID3D12CommandList
An interface from which ID3D12GraphicsCommandList inherits from. It represents an ordered set of commands that the GPU executes, while allowing for extension to support other command lists than just those for graphics (such as compute and copy).
ID3D12CommandQueue
Provides methods for submitting command lists, synchronizing command list execution, instrumenting the command queue, and updating resource tile mappings.
ID3D12CommandSignature
A command signature object enables apps to specify indirect drawing, including the buffer format, command type and resource bindings to be used.
ID3D12DescriptorHeap
A descriptor heap is a collection of contiguous allocations of descriptors, one allocation for every descriptor.
ID3D12Device
Represents a virtual adapter; it is used to create command allocators, command lists, command queues, fences, resources, pipeline state objects, heaps, root signatures, samplers, and many resource views.
ID3D12Device1
Represents a virtual adapter, and expands on the range of methods provided by ID3D12Device.

ID3D12Device10	TBD
ID3D12Device2	Represents a virtual adapter. This interface extends ID3D12Device1 to create pipeline state objects from pipeline state stream descriptions.
ID3D12Device3	Represents a virtual adapter. This interface extends ID3D12Device2 to support the creation of special-purpose diagnostic heaps in system memory that persist even in the event of a GPU-fault or device-removed scenario.
ID3D12Device4	Represents a virtual adapter. This interface extends ID3D12Device3 .
ID3D12Device5	Represents a virtual adapter. This interface extends ID3D12Device4 .
ID3D12Device6	Represents a virtual adapter. This interface extends ID3D12Device5 .
ID3D12Device7	Represents a virtual adapter. This interface extends ID3D12Device6 .
ID3D12Device8	Represents a virtual adapter. This interface extends ID3D12Device7 .
ID3D12Device9	Represents a virtual adapter. This interface extends ID3D12Device8 to add methods to manage shader caches.
ID3D12DeviceChild	An interface from which other core interfaces inherit from, including (but not limited to) ID3D12PipelineLibrary, ID3D12CommandList, ID3D12Pageable, and ID3D12RootSignature. It provides a method to get back to the device object it was created against.
ID3D12DeviceRemovedExtendedData	Provides runtime access to Device Removed Extended Data (DRED) data.

[ID3D12DeviceRemovedExtendedData1](#)

[ID3D12DeviceRemovedExtendedData2](#)

[ID3D12DeviceRemovedExtendedDataSettings](#)

This interface controls Device Removed Extended Data (DRED) settings.

[ID3D12DeviceRemovedExtendedDataSettings1](#)

[ID3D12Fence](#)

Represents a fence, an object used for synchronization of the CPU and one or more GPUs.
([ID3D12Fence](#))

[ID3D12Fence1](#)

Represents a fence. This interface extends [ID3D12Fence](#), and supports the retrieval of the flags used to create the original fence.

[ID3D12GraphicsCommandList](#)

Encapsulates a list of graphics commands for rendering. Includes APIs for instrumenting the command list execution, and for setting and clearing the pipeline state.

[ID3D12GraphicsCommandList1](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support programmable sample positions, atomic copies for implementing late-latch techniques, and optional depth-bounds testing.

[ID3D12GraphicsCommandList2](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support writing immediate values directly to a buffer.

[ID3D12GraphicsCommandList3](#)

Encapsulates a list of graphics commands for rendering.

[ID3D12GraphicsCommandList4](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support ray tracing and render passes.

[ID3D12GraphicsCommandList5](#)

Encapsulates a list of graphics commands for rendering, extending the interface to support variable-rate shading (VRS).

[ID3D12GraphicsCommandList6](#)

[ID3D12GraphicsCommandList7](#)

TBD

[ID3D12Heap](#)

A heap is an abstraction of contiguous memory allocation, used to manage physical memory. This heap can be used with ID3D12Resource objects to support placed resources or reserved resources.

[ID3D12Heap1](#)

[ID3D12LifetimeOwner](#)

Represents an application-defined callback used for being notified of lifetime changes of an object.

[ID3D12LifetimeTracker](#)

Represents facilities for controlling the lifetime a lifetime-tracked object.

[ID3D12MetaCommand](#)

Represents a meta command. A meta command is a Direct3D 12 object representing an algorithm that is accelerated by independent hardware vendors (IHVs). It's an opaque reference to a command generator that is implemented by the driver.

[ID3D12Object](#)

An interface from which ID3D12Device and ID3D12DeviceChild inherit from. It provides methods to associate private data and annotate object names.

[ID3D12Pageable](#)

An interface from which many other core interfaces inherit from. It indicates that the object type encapsulates some amount of GPU-accessible memory; but does not strongly indicate whether the application can manipulate the object's residency.

[ID3D12PipelineLibrary](#)

Manages a pipeline library, in particular loading and retrieving individual PSOs.

[ID3D12PipelineLibrary1](#)

Manages a pipeline library. This interface extends ID3D12PipelineLibrary to load PSOs from a pipeline state stream description.

[ID3D12PipelineState](#)

Represents the state of all currently set shaders as well as certain fixed function state objects.

[ID3D12ProtectedResourceSession](#)

Monitors the validity of a protected resource session. (ID3D12ProtectedResourceSession)

[ID3D12ProtectedResourceSession1](#)

Monitors the validity of a protected resource session. (ID3D12ProtectedResourceSession1)

[ID3D12ProtectedSession](#)

Offers base functionality that allows for a consistent way to monitor the validity of a session across the different types of sessions.

[ID3D12QueryHeap](#)

Manages a query heap. A query heap holds an array of queries, referenced by indexes.

[ID3D12Resource](#)

Encapsulates a generalized ability of the CPU and GPU to read and write to physical memory, or heaps. It contains abstractions for organizing and manipulating simple arrays of data as well as multidimensional data optimized for shader sampling.

[ID3D12Resource1](#)

[ID3D12Resource2](#)

[ID3D12RootSignature](#)

The root signature defines what resources are bound to the graphics pipeline. A root signature is configured by the app and links command lists to the resources the shaders require. Currently, there is one graphics and one compute root signature per app.

[ID3D12RootSignatureDeserializer](#)

Contains a method to return the deserialized D3D12_ROOT_SIGNATURE_DESC data structure, of a serialized root signature version 1.0.

[ID3D12SDKConfiguration](#)

Provides SDK configuration methods.

[ID3D12ShaderCacheSession](#)

Represents a shader cache session.

[ID3D12StateObject](#)

Represents a variable amount of configuration state, including shaders, that an application manages as a single unit and which is given to a driver atomically to process, such as compile or optimize.

[ID3D12StateObjectProperties](#)

Provides methods for getting and setting the properties of an ID3D12StateObject.

[ID3D12SwapChainAssistant](#)

[ID3D12Tools](#)

This interface is used to configure the runtime for tools such as PIX. Its not intended or supported for any other scenario.

[ID3D12VersionedRootSignatureDeserializer](#)

Contains methods to return the deserialized D3D12_ROOT_SIGNATURE_DESC1 data structure, of any version of a serialized root signature.

[ID3D12VirtualizationGuestDevice](#)

TBD

Functions

[D3D12_DECODE_FILTER_REDUCTION](#)

[D3D12_DECODE_IS_ANISOTROPIC_FILTER](#)

[D3D12_DECODE_IS_COMPARISON_FILTER](#)

[D3D12_DECODE_MAG_FILTER](#)

[D3D12_DECODE_MIN_FILTER](#)

[D3D12_DECODE_MIP_FILTER](#)

[D3D12_DECODE_SHADER_4_COMPONENT_MAPPING](#)

[D3D12_ENCODE_ANISOTROPIC_FILTER](#)

[D3D12_ENCODE_BASIC_FILTER](#)

[D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING](#)

[D3D12_GET_COARSE_SHADING_RATE_X_AXIS](#)

[D3D12_GET_COARSE_SHADING_RATE_Y_AXIS](#)

[D3D12_MAKE_COARSE_SHADING_RATE](#)

[D3D12CreateDevice](#)

Creates a device that represents the display adapter. (D3D12CreateDevice)

[D3D12CreateRootSignatureDeserializer](#)

Deserializes a root signature so you can determine the layout definition (D3D12_ROOT_SIGNATURE_DESC).

[D3D12CreateVersionedRootSignatureDeserializer](#)

Generates an interface that can return the deserialized data structure, via GetUnconvertedRootSignatureDesc.

[D3D12EnableExperimentalFeatures](#)

Enables a list of experimental features.

[D3D12GetDebugInterface](#)

Gets a debug interface.

[D3D12GetInterface](#)

Selects an SDK version at runtime when the system is in Windows Developer Mode.

[D3D12SerializeRootSignature](#)

Serializes a root signature version 1.0 that can be passed to ID3D12Device::CreateRootSignature.

[D3D12SerializeVersionedRootSignature](#)

Serializes a root signature of any version that can be passed to ID3D12Device::CreateRootSignature.

Callback functions

[PFN_D3D12_CREATE_DEVICE](#)

[PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER](#)

[PFN_D3D12_CREATE_VERSIONED_ROOT_SIGNATURE_DESERIALIZER](#)

[PFN_D3D12_GET_DEBUG_INTERFACE](#)

[PFN_D3D12_GET_INTERFACE](#)

[PFN_D3D12_SERIALIZE_ROOT_SIGNATURE](#)

[PFN_D3D12_SERIALIZE_VERSIONED_ROOT_SIGNATURE](#)

Structures

[D3D12_AUTO_BREADCRUMB_NODE](#)

Represents Device Removed Extended Data (DRED) auto-breadcrumb data as a node in a linked list.

[D3D12_AUTO_BREADCRUMB_NODE1](#)

[D3D12_BARRIER_GROUP](#)

Describes a group of barriers of a given type.

[D3D12_BARRIER_SUBRESOURCE_RANGE](#)

Allows you to transition logically-adjacent ranges of subresources.

[D3D12_BLEND_DESC](#)

Describes the blend state. (D3D12_BLEND_DESC)

[D3D12_BOX](#)

Describes a 3D box.

[D3D12_BUFFER_BARRIER](#)

Describes a buffer memory access barrier. Used by buffer barriers to indicate when resource memory must be made visible for a specific access type.

[D3D12_BUFFER_RTV](#)

Describes the elements in a buffer resource to use in a render-target view.

[D3D12_BUFFER_SRV](#)

Describes the elements in a buffer resource to use in a shader-resource view.

[D3D12_BUFFER_UAV](#)

Describes the elements in a buffer to use in a unordered-access view. (D3D12_BUFFER_UAV)

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC](#)

Describes a raytracing acceleration structure. Pass this structure into ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure to describe the acceleration structure to be built.

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#)

Defines the inputs for a raytracing acceleration structure build operation. This structure is used by ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure and ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo.

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER](#)

Describes the GPU memory layout of an acceleration structure visualization.

[D3D12_CACHED_PIPELINE_STATE](#)

Stores a pipeline state.

[D3D12_CLEAR_VALUE](#)

Describes a value used to optimize clear operations for a particular resource.

[D3D12_COMMAND_QUEUE_DESC](#)

Describes a command queue.

[D3D12_COMMAND_SIGNATURE_DESC](#)

Describes the arguments (parameters) of a command signature.

[D3D12_COMPUTE_PIPELINE_STATE_DESC](#)

Describes a compute pipeline state object.

[D3D12_CONSTANT_BUFFER_VIEW_DESC](#)

Describes a constant buffer to view.

[D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes a CPU descriptor handle.

[D3D12_DEPTH_STENCIL_DESC](#)

Describes depth-stencil state. (D3D12_DEPTH_STENCIL_DESC)

[D3D12_DEPTH_STENCIL_DESC1](#)

Describes depth-stencil state. (D3D12_DEPTH_STENCIL_DESC1)

[D3D12_DEPTH_STENCIL_VALUE](#)

Specifies a depth and stencil value.

[D3D12_DEPTH_STENCIL_VIEW_DESC](#)

	Describes the subresources of a texture that are accessible from a depth-stencil view.
D3D12_DEPTH_STENCILOP_DESC	Describes stencil operations that can be performed based on the results of stencil test.
D3D12_DESCRIPTOR_HEAP_DESC	Describes the descriptor heap.
D3D12_DESCRIPTOR_RANGE	Describes a descriptor range.
D3D12_DESCRIPTOR_RANGE1	Describes a descriptor range, with flags to determine their volatility.
D3D12_DEVICE_REMOVED_EXTENDED_DATA	Represents Device Removed Extended Data (DRED) version 1.0 data.
D3D12_DEVICE_REMOVED_EXTENDED_DATA1	Represents Device Removed Extended Data (DRED) version 1.1 data.
D3D12_DEVICE_REMOVED_EXTENDED_DATA2	
D3D12_DEVICE_REMOVED_EXTENDED_DATA3	
D3D12_DISCARD_REGION	Describes details for the discard-resource operation.
D3D12_DISPATCH_ARGUMENTS	Describes dispatch parameters, for use by the compute shader.
D3D12_DISPATCH_MESH_ARGUMENTS	
D3D12_DISPATCH_RAYS_DESC	Describes the properties of a ray dispatch operation initiated with a call to ID3D12GraphicsCommandList4::DispatchRays.
D3D12_DRAW_ARGUMENTS	

	Describes parameters for drawing instances.
	D3D12_DRAW_INDEXED_ARGUMENTS
	Describes parameters for drawing indexed instances.
	D3D12_DRED_ALLOCATION_NODE
	Describes, as a node in a linked list, data about an allocation tracked by Device Removed Extended Data (DRED).
	D3D12_DRED_ALLOCATION_NODE1
	D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT
	Contains a pointer to the head of a linked list of D3D12_AUTO_BREADCRUMB_NODE objects.
	D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1
	D3D12_DRED_BREADCRUMB_CONTEXT
	D3D12_DRED_PAGE_FAULT_OUTPUT
	Describes allocation data related to a GPU page fault on a given virtual address (VA).
	D3D12_DRED_PAGE_FAULT_OUTPUT1
	D3D12_DRED_PAGE_FAULT_OUTPUT2
	D3D12_DXIL_LIBRARY_DESC
	Describes a DXIL library state subobject that can be included in a state object.
	D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION
	This subobject is unsupported in the current release.
	D3D12_EXISTING_COLLECTION_DESC
	A state subobject describing an existing collection that can be included in a state object.
	D3D12_EXPORT_DESC
	Describes an export from a state subobject such as a DXIL library or a collection state object.

[D3D12_FEATURE_DATA_ARCHITECTURE](#)

Provides detail about the adapter architecture, so that your application can better optimize for certain adapter properties.

[D3D12_FEATURE_DATA_ARCHITECTURE1](#)

Provides detail about each adapter's architectural details, so that your application can better optimize for certain adapter properties.

[D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY](#)

Details the adapter's support for prioritization of different command queue types.

[D3D12_FEATURE_DATA_CROSS_NODE](#)

Indicates the level of support for the sharing of resources between different adapters—for example, multiple GPUs.

[D3D12_FEATURE_DATA_D3D12_OPTIONS](#)

Describes Direct3D 12 feature options in the current graphics driver.

[D3D12_FEATURE_DATA_D3D12_OPTIONS1](#)

Describes the level of support for HLSL 6.0 wave operations.

[D3D12_FEATURE_DATA_D3D12_OPTIONS10](#)

Indicates whether or not the SUM combiner can be used, and whether or not *SV_ShadingRate* can be set from a mesh shader.

[D3D12_FEATURE_DATA_D3D12_OPTIONS11](#)

Indicates whether or not 64-bit integer atomics on resources in descriptor heaps are supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS12](#)

Indicates whether or not Enhanced Barriers are supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS13](#)

TBD

[D3D12_FEATURE_DATA_D3D12_OPTIONS2](#)

Indicates the level of support that the adapter provides for depth-bounds tests and programmable sample positions.

[D3D12_FEATURE_DATA_D3D12_OPTIONS3](#)

Indicates the level of support that the adapter provides for timestamp queries, format-casting, immediate write, view instancing, and barycentrics.

[D3D12_FEATURE_DATA_D3D12_OPTIONS4](#)

Indicates the level of support for 64KB-aligned MSAA textures, cross-API sharing, and native 16-bit shader operations.

[D3D12_FEATURE_DATA_D3D12_OPTIONS5](#)

Indicates the level of support that the adapter provides for render passes, ray tracing, and shader-resource view tier 3 tiled resources.

[D3D12_FEATURE_DATA_D3D12_OPTIONS6](#)

Indicates the level of support that the adapter provides for variable-rate shading (VRS), and indicates whether or not background processing is supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS7](#)

Indicates the level of support that the adapter provides for mesh and amplification shaders, and for sampler feedback.

[D3D12_FEATURE_DATA_D3D12_OPTIONS8](#)

Indicates whether or not unaligned block-compressed textures are supported.

[D3D12_FEATURE_DATA_D3D12_OPTIONS9](#)

Indicates whether or not support exists for mesh shaders, values of *SV_RenderTargetArrayIndex* that are 8 or greater, typed resource 64-bit integer atomics, derivative and derivative-dependent texture sample operations, and the level of support for WaveMMA (wave_matrix) operations.

[D3D12_FEATURE_DATA_DISPLAYABLE](#)

This feature is currently in preview.

[D3D12_FEATURE_DATA_EXISTING_HEAPS](#)

Provides detail about whether the adapter supports creating heaps from existing system memory.

[D3D12_FEATURE_DATA_FEATURE_LEVELS](#)

Describes info about the feature levels supported by the current graphics driver.

[D3D12_FEATURE_DATA_FORMAT_INFO](#)

	Describes a DXGI data format and plane count.
D3D12_FEATURE_DATA_FORMAT_SUPPORT	Describes which resources are supported by the current graphics driver for a given format. (D3D12_FEATURE_DATA_FORMAT_SUPPORT)
D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT	Details the adapter's GPU virtual address space limitations, including maximum address bits per resource and per process.
D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS	Describes the multi-sampling image quality levels for a given format and sample count.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT	Indicates the level of support for protected resource sessions.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT	Indicates a count of protected resource session types.
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES	Indicates a list of protected resource session types.
D3D12_FEATURE_DATA_QUERY_META_COMMAND	Indicates the level of support that the adapter provides for metacommmands.
D3D12_FEATURE_DATA_ROOT_SIGNATURE	Indicates root signature version support.
D3D12_FEATURE_DATA_SERIALIZATION	Indicates the level of support for heap serialization.
D3D12_FEATURE_DATA_SHADER_CACHE	Describes the level of shader caching supported in the current graphics driver. (D3D12_FEATURE_DATA_SHADER_CACHE)
D3D12_FEATURE_DATA_SHADER_MODEL	Contains the supported shader model.

D3D12_GLOBAL_BARRIER	Describes a resource memory access barrier. Used by global, texture, and buffer barriers to indicate when resource memory must be made visible for a specific access type.
D3D12_GLOBAL_ROOT_SIGNATURE	Defines a global root signature state subobject that will be used with associated shaders.
D3D12_GPU_DESCRIPTOR_HANDLE	Describes a GPU descriptor handle.
D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE	Represents a GPU virtual address and indexing stride.
D3D12_GPU_VIRTUAL_ADDRESS_RANGE	Represents a GPU virtual address range.
D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE	Represents a GPU virtual address range and stride.
D3D12_GRAPHICS_PIPELINE_STATE_DESC	Describes a graphics pipeline state object.
D3D12_HEAP_DESC	Describes a heap.
D3D12_HEAP_PROPERTIES	Describes heap properties.
D3D12_HIT_GROUP_DESC	Describes a raytracing hit group state subobject that can be included in a state object.
D3D12_INDEX_BUFFER_VIEW	Describes the index buffer to view.
D3D12_INDIRECT_ARGUMENT_DESC	Describes an indirect argument (an indirect parameter), for use with a command signature.

[D3D12_INPUT_ELEMENT_DESC](#)

Describes a single element for the input-assembler stage of the graphics pipeline.

[D3D12_INPUT_LAYOUT_DESC](#)

Describes the input-buffer data for the input-assembler stage.

[D3D12_LOCAL_ROOT_SIGNATURE](#)

Defines a local root signature state subobject that will be used with associated shaders.

[D3D12_MEMCPY_DEST](#)

Describes the destination of a memory copy operation.

[D3D12_META_COMMAND_DESC](#)

Describes a meta command.

[D3D12_META_COMMAND_PARAMETER_DESC](#)

Describes a parameter to a meta command.

[D3D12_MIP_REGION](#)

Describes the dimensions of a mip region.

[D3D12_NODE_MASK](#)

A state subobject that identifies the GPU nodes to which the state object applies.

[D3D12_PACKED_MIP_INFO](#)

Describes the tile structure of a tiled resource with mipmaps. (D3D12_PACKED_MIP_INFO)

[D3D12_PIPELINE_STATE_STREAM_DESC](#)

Describes a pipeline state stream.

[D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#)

Describes the footprint of a placed subresource, including the offset and the D3D12_SUBRESOURCE_FOOTPRINT.

[D3D12_PROTECTED_RESOURCE_SESSION_DESC](#)

Describes flags for a protected resource session, per adapter.

[D3D12_PROTECTED_RESOURCE_SESSION_DESC1](#)

Describes flags and protection type for a protected resource session, per adapter.

[D3D12_QUERY_DATA_PIPELINE_STATISTICS](#)

Query information about graphics-pipeline activity in between calls to BeginQuery and EndQuery.

[D3D12_QUERY_DATA_PIPELINE_STATISTICS1](#)

[D3D12_QUERY_DATA_SO_STATISTICS](#)

Describes query data for stream output.

[D3D12_QUERY_HEAP_DESC](#)

Describes the purpose of a query heap. A query heap contains an array of individual queries.

[D3D12_RANGE](#)

Describes a memory range.

[D3D12_RANGE_UINT64](#)

Describes a memory range in a 64-bit address space.

[D3D12_RASTERIZER_DESC](#)

Describes rasterizer state. (D3D12_RASTERIZER_DESC)

[D3D12_RAYTRACING_AABB](#)

Represents an axis-aligned bounding box (AABB) used as raytracing geometry.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC](#)

Describes the space requirement for acceleration structure after compaction.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC](#)

Describes the space currently used by an acceleration structure..

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC](#)

Description of the post-build information to generate from an acceleration structure. Use this structure in calls to `EmitRaytracingAccelerationStructurePostbuildInfo` and `BuildRaytracingAccelerationStructure`.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC](#)

Describes the size and layout of the serialized acceleration structure and header.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC](#)

Describes the space requirement for decoding an acceleration structure into a form that can be visualized by tools.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO](#)

Represents prebuild information about a raytracing acceleration structure. Get an instance of this structure by calling `GetRaytracingAccelerationStructurePrebuildInfo`.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV](#)

A shader resource view (SRV) structure for storing a raytracing acceleration structure.

[D3D12_RAYTRACING_GEOMETRY_AABBS_DESC](#)

Describes a set of Axis-aligned bounding boxes that are used in the `D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS` structure to provide input data to a raytracing acceleration structure build operation.

[D3D12_RAYTRACING_GEOMETRY_DESC](#)

Describes a set of geometry that is used in the `D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS` structure to provide input data to a raytracing acceleration structure build operation.

[D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC](#)

Describes a set of triangles used as raytracing geometry. The geometry pointed to by this struct are always in triangle list form, indexed or non-indexed. Triangle strips are not supported.

[D3D12_RAYTRACING_INSTANCE_DESC](#)

Describes an instance of a raytracing acceleration structure used in GPU memory during the acceleration structure build process.

[D3D12_RAYTRACING_PIPELINE_CONFIG](#)

A state subobject that represents a raytracing pipeline configuration.

[D3D12_RAYTRACING_PIPELINE_CONFIG1](#)

A state subobject that represents a raytracing pipeline configuration, with flags.

[D3D12_RAYTRACING_SHADER_CONFIG](#)

A state subobject that represents a shader configuration.

[D3D12_RENDER_PASS_BEGINNING_ACCESS](#)

Describes the access to resource(s) that is requested by an application at the transition into a render pass.

[D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS](#)

Describes the clear value to which resource(s) should be cleared at the beginning of a render pass.

[D3D12_RENDER_PASS_DEPTH_STENCIL_DESC](#)

Describes a binding (fixed for the duration of the render pass) to a depth stencil view (DSV), as well as its beginning and ending access characteristics.

[D3D12_RENDER_PASS_ENDING_ACCESS](#)

Describes the access to resource(s) that is requested by an application at the transition out of a render pass.

[D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS](#)

Describes a resource to resolve to at the conclusion of a render pass.

[D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS](#)

Describes the subresources involved in resolving at the conclusion of a render pass.

[D3D12_RENDER_PASS_RENDER_TARGET_DESC](#)

Describes bindings (fixed for the duration of the render pass) to one or more render target views (RTVs), as well as their beginning and ending access characteristics.

[D3D12_RENDER_TARGET_BLEND_DESC](#)

Describes the blend state for a render target. (D3D12_RENDER_TARGET_BLEND_DESC)

[D3D12_RENDER_TARGET_VIEW_DESC](#)

Describes the subresources from a resource that are accessible by using a render-target view.

[D3D12_RESOURCE_ALIASING_BARRIER](#)

Describes the transition between usages of two different resources that have mappings into the same heap.

[D3D12_RESOURCE_ALLOCATION_INFO](#)

Describes parameters needed to allocate resources.

[D3D12_RESOURCE_ALLOCATION_INFO1](#)

Describes parameters needed to allocate resources, including offset.

[D3D12_RESOURCE_BARRIER](#)

Describes a resource barrier (transition in resource use).

[D3D12_RESOURCE_DESC](#)

Describes a resource, such as a texture. This structure is used extensively.

[D3D12_RESOURCE_DESC1](#)

Describes a resource, such as a texture, including a mip region. This structure is used in several methods.

[D3D12_RESOURCE_TRANSITION_BARRIER](#)

Describes the transition of subresources between different usages.

[D3D12_RESOURCE_UAV_BARRIER](#)

Represents a resource in which all UAV accesses must complete before any future UAV accesses can begin.

[D3D12_ROOT_CONSTANTS](#)

Describes constants inline in the root signature that appear in shaders as one constant buffer.

[D3D12_ROOT_DESCRIPTOR](#)

Describes descriptors inline in the root signature version 1.0 that appear in shaders.

[D3D12_ROOT_DESCRIPTOR_TABLE](#)

Describes the root signature 1.0 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

[D3D12_ROOT_DESCRIPTOR_TABLE1](#)

Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

[D3D12_ROOT_DESCRIPTOR1](#)

Describes descriptors inline in the root signature version 1.1 that appear in shaders.

[D3D12_ROOT_PARAMETER](#)

Describes the slot of a root signature version 1.0.

[D3D12_ROOT_PARAMETER1](#)

Describes the slot of a root signature version 1.1.

[D3D12_ROOT_SIGNATURE_DESC](#)

Describes the layout of a root signature version 1.0.

[D3D12_ROOT_SIGNATURE_DESC1](#)

Describes the layout of a root signature version 1.1.

[D3D12_RT_FORMAT_ARRAY](#)

Wraps an array of render target formats.

[D3D12_SAMPLE_POSITION](#)

Describes a sub-pixel sample position for use with programmable sample positions.

[D3D12_SAMPLER_DESC](#)

Describes a sampler state. (D3D12_SAMPLER_DESC)

[D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER](#)

Opaque data structure describing driver versioning for a serialized acceleration structure.

[D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER](#)

Defines the header for a serialized raytracing acceleration structure.

[D3D12_SHADER_BYTECODE](#)

Describes shader data. (D3D12_SHADER_BYTECODE)

[D3D12_SHADER_CACHE_SESSION_DESC](#)

Describes a shader cache session.

[D3D12_SHADER_RESOURCE_VIEW_DESC](#)

Describes a shader-resource view. (D3D12_SHADER_RESOURCE_VIEW_DESC)
D3D12_SO_DECLARATION_ENTRY
Describes a vertex element in a vertex buffer in an output slot.
D3D12_STATE_OBJECT_CONFIG
Defines general properties of a state object.
D3D12_STATE_OBJECT_DESC
Description of a state object. Pass this structure into ID3D12Device::CreateStateObject.
D3D12_STATE_SUBOBJECT
Represents a subobject within a state object description. Use with D3D12_STATE_OBJECT_DESC .
D3D12_STATIC_SAMPLER_DESC
Describes a static sampler.
D3D12_STREAM_OUTPUT_BUFFER_VIEW
Describes a stream output buffer.
D3D12_STREAM_OUTPUT_DESC
Describes a streaming output buffer.
D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION
Associates a subobject defined directly in a state object with shader exports.
D3D12_SUBRESOURCE_DATA
Describes subresource data. (D3D12_SUBRESOURCE_DATA)
D3D12_SUBRESOURCE_FOOTPRINT
Describes the format, width, height, depth, and row-pitch of the subresource into the parent resource.
D3D12_SUBRESOURCE_INFO
Describes subresource data. (D3D12_SUBRESOURCE_INFO)
D3D12_SUBRESOURCE_RANGE_UINT64
Describes a subresource memory range.

[D3D12_SUBRESOURCE_TILING](#)

Describes a tiled subresource volume. (D3D12_SUBRESOURCE_TILING)

[D3D12_TEX1D_ARRAY_DSV](#)

Describes the subresources from an array of 1D textures to use in a depth-stencil view.

[D3D12_TEX1D_ARRAY_RTV](#)

Describes the subresources from an array of 1D textures to use in a render-target view.

[D3D12_TEX1D_ARRAY_SRV](#)

Describes the subresources from an array of 1D textures to use in a shader-resource view.

[D3D12_TEX1D_ARRAY_UAV](#)

Describes an array of unordered-access 1D texture resources. (D3D12_TEX1D_ARRAY_UAV)

[D3D12_TEX1D_DSV](#)

Describes the subresource from a 1D texture that is accessible to a depth-stencil view.

[D3D12_TEX1D_RTV](#)

Describes the subresource from a 1D texture to use in a render-target view.

[D3D12_TEX1D_SRV](#)

Specifies the subresource from a 1D texture to use in a shader-resource view. (D3D12_TEX1D_SRV)

[D3D12_TEX1D_UAV](#)

Describes a unordered-access 1D texture resource. (D3D12_TEX1D_UAV)

[D3D12_TEX2D_ARRAY_DSV](#)

Describes the subresources from an array of 2D textures that are accessible to a depth-stencil view.

[D3D12_TEX2D_ARRAY_RTV](#)

Describes the subresources from an array of 2D textures to use in a render-target view.
(D3D12_TEX2D_ARRAY_RTV)

[D3D12_TEX2D_ARRAY_SRV](#)

Describes the subresources from an array of 2D textures to use in a shader-resource view.
(D3D12_TEX2D_ARRAY_SRV)

[D3D12_TEX2D_ARRAY_UAV](#)

Describes an array of unordered-access 2D texture resources. (D3D12_TEX2D_ARRAY_UAV)

[D3D12_TEX2D_DSV](#)

Describes the subresource from a 2D texture that is accessible to a depth-stencil view.

[D3D12_TEX2D_RTV](#)

Describes the subresource from a 2D texture to use in a render-target view. (D3D12_TEX2D_RTV)

[D3D12_TEX2D_SRV](#)

Describes the subresource from a 2D texture to use in a shader-resource view. (D3D12_TEX2D_SRV)

[D3D12_TEX2D_UAV](#)

Describes a unordered-access 2D texture resource. (D3D12_TEX2D_UAV)

[D3D12_TEX2DMS_ARRAY_DSV](#)

Describes the subresources from an array of multi sampled 2D textures for a depth-stencil view.

[D3D12_TEX2DMS_ARRAY_RTV](#)

Describes the subresources from an array of multi sampled 2D textures to use in a render-target view.

[D3D12_TEX2DMS_ARRAY_SRV](#)

Describes the subresources from an array of multi sampled 2D textures to use in a shader-resource view.

[D3D12_TEX2DMS_DSV](#)

Describes the subresource from a multi sampled 2D texture that is accessible to a depth-stencil view.

[D3D12_TEX2DMS_RTV](#)

Describes the subresource from a multi sampled 2D texture to use in a render-target view.

[D3D12_TEX2DMS_SRV](#)

Describes the subresources from a multi sampled 2D texture to use in a shader-resource view.

[D3D12_TEX3D_RTV](#)

	Describes the subresources from a 3D texture to use in a render-target view.
D3D12_TEX3D_SRV	Describes the subresources from a 3D texture to use in a shader-resource view.
D3D12_TEX3D_UAV	Describes a unordered-access 3D texture resource. (D3D12_TEX3D_UAV)
D3D12_TEXCUBE_ARRAY_SRV	Describes the subresources from an array of cube textures to use in a shader-resource view.
D3D12_TEXCUBE_SRV	Describes the subresource from a cube texture to use in a shader-resource view.
D3D12_TEXTURE_BARRIER	Expresses an access transition for a texture.
D3D12_TEXTURE_COPY_LOCATION	Describes a portion of a texture for the purpose of texture copies.
D3D12_TILE_REGION_SIZE	Describes the size of a tiled region. (D3D12_TILE_REGION_SIZE)
D3D12_TILE_SHAPE	Describes the shape of a tile by specifying its dimensions. (D3D12_TILE_SHAPE)
D3D12_TILED_RESOURCE_COORDINATE	Describes the coordinates of a tiled resource. (D3D12_TILED_RESOURCE_COORDINATE)
D3D12_UNORDERED_ACCESS_VIEW_DESC	Describes the subresources from a resource that are accessible by using an unordered-access view.
D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA	Represents versioned Device Removed Extended Data (DRED) data.
D3D12_VERSIONED_ROOT_SIGNATURE_DESC	Holds any version of a root signature description, and is designed to be used with serialization/deserialization functions.

D3D12_VERTEX_BUFFER_VIEW

Describes a vertex buffer view.

D3D12_VIEW_INSTANCE_LOCATION

Specifies the viewport/stencil and render target associated with a view instance.

D3D12_VIEW_INSTANCING_DESC

Specifies parameters used during view instancing configuration.

D3D12_VIEWPORT

Describes the dimensions of a viewport.

D3D12_WRITEBUFFERIMMEDIATE_PARAMETER

Specifies the immediate value and destination address written using ID3D12CommandList2::WriteBufferImmediate.

Enumerations

D3D_ROOT_SIGNATURE_VERSION

Specifies the version of root signature layout.

D3D_SHADER_MODEL

Specifies a shader model.

D3D12_AUTO_BREADCRUMB_OP

Defines constants that specify render/compute GPU operations.
(D3D12_AUTO_BREADCRUMB_OP)

D3D12_AXIS_SHADING_RATE

Defines constants that specify the shading rate (for variable-rate shading, or VRS) along a horizontal or vertical axis.

D3D12_BACKGROUND_PROCESSING_MODE

Defines constants that specify a level of dynamic optimization to apply to GPU work that's subsequently submitted.

[D3D12_BARRIER_ACCESS](#)

[D3D12_BARRIER_LAYOUT](#)

[D3D12_BARRIER_SYNC](#)

[D3D12_BARRIER_TYPE](#)

[D3D12_BLEND](#)

Specifies blend factors, which modulate values for the pixel shader and render target.

[D3D12_BLEND_OP](#)

Specifies RGB or alpha blending operations.

[D3D12_BUFFER_SRV_FLAGS](#)

Identifies how to view a buffer resource. (D3D12_BUFFER_SRV_FLAGS)

[D3D12_BUFFER_UAV_FLAGS](#)

Identifies unordered-access view options for a buffer resource. (D3D12_BUFFER_UAV_FLAGS)

[D3D12_CLEAR_FLAGS](#)

Specifies what to clear from the depth stencil view.

[D3D12_COLOR_WRITE_ENABLE](#)

Identifies which components of each pixel of a render target are writable during blending.

[D3D12_COMMAND_LIST_FLAGS](#)

The D3D12_COMMAND_LIST_FLAGS enumeration specifies flags to be used when creating a command list.

[D3D12_COMMAND_LIST_SUPPORT_FLAGS](#)

Used to determine which kinds of command lists are capable of supporting various operations.

[D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of a command list.

[D3D12_COMMAND_POOL_FLAGS](#)

[D3D12_COMMAND_QUEUE_FLAGS](#)

Specifies flags to be used when creating a command queue.

[D3D12_COMMAND_QUEUE_PRIORITY](#)

Defines priority levels for a command queue.

[D3D12_COMMAND_RECORDER_FLAGS](#)

[D3D12_COMPARISON_FUNC](#)

Specifies comparison options.

[D3D12_CONSERVATIVE_RASTERIZATION_MODE](#)

Identifies whether conservative rasterization is on or off.

([D3D12_CONSERVATIVE_RASTERIZATION_MODE](#))

[D3D12_CONSERVATIVE_RASTERIZATION_TIER](#)

Identifies the tier level of conservative rasterization.

[D3D12_CPU_PAGE_PROPERTY](#)

Specifies the CPU-page properties for the heap.

[D3D12_CROSS_NODE_SHARING_TIER](#)

Specifies the level of sharing across nodes of an adapter, such as Tier 1 Emulated, Tier 1, or Tier 2.

[D3D12_CULL_MODE](#)

Specifies triangles facing a particular direction are not drawn.

[D3D12_DEPTH_WRITE_MASK](#)

Identifies the portion of a depth-stencil buffer for writing depth data.

[D3D12_DESCRIPTOR_HEAP_FLAGS](#)

Specifies options for a heap.

[D3D12_DESCRIPTOR_HEAP_TYPE](#)

Specifies a type of descriptor heap.
D3D12_DESCRIPTOR_RANGE_FLAGS
Specifies the volatility of both descriptors and the data they reference in a Root Signature 1.1 description, which can enable some driver optimizations.
D3D12_DESCRIPTOR_RANGE_TYPE
Specifies a range so that, for example, if part of a descriptor table has 100 shader-resource views (SRVs) that range can be declared in one entry rather than 100.
D3D12_DRED_ALLOCATION_TYPE
Congruent with, and numerically equivalent to, 3D12DDI_HANDLETYPE enumeration values.
D3D12_DRED_DEVICE_STATE
D3D12_DRED_ENABLEMENT
Defines constants that specify render/compute GPU operations. (D3D12_DRED_ENABLEMENT)
D3D12_DRED_FLAGS
Defines constants used in the D3D12_DEVICE_REMOVED_EXTENDED_DATA structure to specify control flags for the Direct3D runtime.
D3D12_DRED_PAGE_FAULT_FLAGS
D3D12_DRED_VERSION
Defines constants that specify a version of Device Removed Extended Data (DRED), as used by the D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure.
D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS
Specifies the result of a call to ID3D12Device5::CheckDriverMatchingIdentifier which queries whether serialized data is compatible with the current device and driver version.
D3D12_DSV_DIMENSION
Specifies how to access a resource used in a depth-stencil view. (D3D12_DSV_DIMENSION)
D3D12_DSV_FLAGS
Specifies depth-stencil view options.

D3D12_ELEMENTS_LAYOUT	Describes how the locations of elements are identified.
D3D12_EXPORT_FLAGS	The flags to apply when exporting symbols from a state subobject.
D3D12_FEATURE	Defines constants that specify a Direct3D 12 feature or feature set to query about.
D3D12_FENCE_FLAGS	Specifies fence options. (D3D12_FENCE_FLAGS)
D3D12_FILL_MODE	Specifies the fill mode to use when rendering triangles.
D3D12_FILTER	Specifies filtering options during texture sampling.
D3D12_FILTER_REDUCTION_TYPE	Specifies the type of filter reduction.
D3D12_FILTER_TYPE	Specifies the type of magnification or minification sampler filters.
D3D12_FORMAT_SUPPORT1	Specifies resources that are supported for a provided format.
D3D12_FORMAT_SUPPORT2	Specifies which unordered resource options are supported for a provided format.
D3D12_GRAPHICS_STATES	Defines flags that specify states related to a graphics command list. Values can be bitwise OR'd together.
D3D12_HEAP_FLAGS	Specifies heap options, such as whether the heap can contain textures, and whether resources are shared across adapters.

[D3D12_HEAP_SERIALIZATION_TIER](#)

Defines constants that specify heap serialization support.

[D3D12_HEAP_TYPE](#)

Specifies the type of heap. When resident, heaps reside in a particular physical memory pool with certain CPU cache properties.

[D3D12_HIT_GROUP_TYPE](#)

Specifies the type of a raytracing hit group state subobject. Use a value from this enumeration with the D3D12_HIT_GROUP_DESC structure.

[D3D12_HIT_KIND](#)

[D3D12_INDEX_BUFFER_STRIP_CUT_VALUE](#)

When using triangle strip primitive topology, vertex positions are interpreted as vertices of a continuous triangle "strip".

[D3D12_INDIRECT_ARGUMENT_TYPE](#)

Specifies the type of the indirect parameter.

[D3D12_INPUT_CLASSIFICATION](#)

Identifies the type of data contained in an input slot.

[D3D12_LIFETIME_STATE](#)

Defines constants that specify the lifetime state of a lifetime-tracked object.

[D3D12_LOGIC_OP](#)

Specifies logical operations to configure for a render target. (D3D12_LOGIC_OP)

[D3D12_MEASUREMENTS_ACTION](#)

Defines constants that specify what should be done with the results of earlier workload instrumentation.

[D3D12_MEMORY_POOL](#)

Specifies the memory pool for the heap.

[D3D12_MESH_SHADER_TIER](#)

Defines constants that specify mesh and amplification shader support.

[D3D12_META_COMMAND_PARAMETER_FLAGS](#)

Defines constants that specify the flags for a parameter to a meta command. Values can be bitwise OR'd together.

[D3D12_META_COMMAND_PARAMETER_STAGE](#)

Defines constants that specify the stage of a parameter to a meta command.

[D3D12_META_COMMAND_PARAMETER_TYPE](#)

Defines constants that specify the data type of a parameter to a meta command.

[D3D12_MULTIPLE_FENCE_WAIT_FLAGS](#)

Specifies multiple wait flags for multiple fences.

[D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS](#)

Specifies options for determining quality levels.

[D3D12_PIPELINE_STATE_FLAGS](#)

Flags to control pipeline state.

[D3D12_PIPELINE_STATE_SUBOBJECT_TYPE](#)

Specifies the type of a sub-object in a pipeline state stream description.

[D3D12_PREDICATION_OP](#)

Specifies the predication operation to apply.

[D3D12_PRIMITIVE_TOPOLOGY_TYPE](#)

Specifies how the pipeline interprets geometry or hull shader input primitives.

[D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER](#)

Specifies the level of support for programmable sample positions that's offered by the adapter.

[D3D12_PROTECTED_RESOURCE_SESSION_FLAGS](#)

Defines constants that specify protected resource session flags.

[D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAGS](#)

Defines constants that specify protected resource session support.

D3D12_PROTECTED_SESSION_STATUS	Defines constants that specify protected session status.
D3D12_QUERY_HEAP_TYPE	Specifies the type of query heap to create.
D3D12_QUERY_TYPE	Specifies the type of query.
D3D12_RAY_FLAGS	Flags passed to the TraceRay function to override transparency, culling, and early-out behavior.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS	Specifies flags for the build of a raytracing acceleration structure. Use a value from this enumeration with the D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS structure that provides input to the acceleration structure build operation.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE	Specifies the type of copy operation performed when calling CopyRaytracingAccelerationStructure.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE	Specifies the type of acceleration structure post-build info that can be retrieved with calls to EmitRaytracingAccelerationStructurePostbuildInfo and BuildRaytracingAccelerationStructure.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE	Specifies the type of a raytracing acceleration structure.
D3D12_RAYTRACING_GEOMETRY_FLAGS	Specifies flags for raytracing geometry in a D3D12_RAYTRACING_GEOMETRY_DESC structure.
D3D12_RAYTRACING_GEOMETRY_TYPE	Specifies the type of geometry used for raytracing. Use a value from this enumeration to specify the geometry type in a D3D12_RAYTRACING_GEOMETRY_DESC.
D3D12_RAYTRACING_INSTANCE_FLAGS	Flags for a raytracing acceleration structure instance. These flags can be used to override D3D12_RAYTRACING_GEOMETRY_FLAGS for individual instances.

D3D12_RAYTRACING_PIPELINE_FLAGS	Defines constants that specify configuration flags for a raytracing pipeline.
D3D12_RAYTRACING_TIER	Specifies the level of ray tracing support on the graphics device.
D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE	Specifies the type of access that an application is given to the specified resource(s) at the transition into a render pass.
D3D12_RENDER_PASS_ENDING_ACCESS_TYPE	Specifies the type of access that an application is given to the specified resource(s) at the transition out of a render pass.
D3D12_RENDER_PASS_FLAGS	Specifies the nature of the render pass; for example, whether it is a suspending or a resuming render pass.
D3D12_RENDER_PASS_TIER	Specifies the level of support for render passes on a graphics device.
D3D12_RESIDENCY_FLAGS	Used with the EnqueueMakeResident function to choose how residency operations proceed when the memory budget is exceeded.
D3D12_RESIDENCY_PRIORITY	Specifies broad residency priority buckets useful for quickly establishing an application priority scheme.
D3D12_RESOLVE_MODE	Specifies a resolve operation.
D3D12_RESOURCE_BARRIER_FLAGS	Flags for setting split resource barriers.
D3D12_RESOURCE_BARRIER_TYPE	Specifies a type of resource barrier (transition in resource use) description.

D3D12_RESOURCE_BINDING_TIER	Identifies the tier of resource binding being used.
D3D12_RESOURCE_DIMENSION	Identifies the type of resource being used. (D3D12_RESOURCE_DIMENSION)
D3D12_RESOURCE_FLAGS	Specifies options for working with resources.
D3D12_RESOURCE_HEAP_TIER	Specifies which resource heap tier the hardware and driver support.
D3D12_RESOURCE_STATES	Defines constants that specify the state of a resource regarding how the resource is being used.
D3D12_ROOT_DESCRIPTOR_FLAGS	Specifies the volatility of the data referenced by descriptors in a Root Signature 1.1 description, which can enable some driver optimizations.
D3D12_ROOT_PARAMETER_TYPE	Specifies the type of root signature slot.
D3D12_ROOT_SIGNATURE_FLAGS	Specifies options for root signature layout.
D3D12_RTV_DIMENSION	Identifies the type of resource to view as a render target.
D3D12_SAMPLER_FEEDBACK_TIER	Defines constants that specify sampler feedback support.
D3D12_SERIALIZED_DATA_TYPE	Specifies the type of serialized data. Use a value from this enumeration when calling ID3D12Device5::CheckDriverMatchingIdentifier.
D3D12_SHADER_CACHE_CONTROL_FLAGS	Defines constants that specify shader cache control options.

D3D12_SHADER_CACHE_FLAGS	Defines constants that specify shader cache flags.
D3D12_SHADER_CACHE_KIND_FLAGS	Defines constants that specify a kind of shader cache.
D3D12_SHADER_CACHE_MODE	Defines constants that specify a shader cache's mode.
D3D12_SHADER_CACHE_SUPPORT_FLAGS	Describes the level of support for shader caching in the current graphics driver. (D3D12_SHADER_CACHE_SUPPORT_FLAGS)
D3D12_SHADER_COMPONENT_MAPPING	Specifies how memory gets routed by a shader resource view (SRV).
D3D12_SHADER_MIN_PRECISION_SUPPORT	Describes minimum precision support options for shaders in the current graphics driver.
D3D12_SHADER_VISIBILITY	Specifies the shaders that can access the contents of a given root signature slot.
D3D12_SHADING_RATE	Defines constants that specify the shading rate (for variable-rate shading, or VRS).
D3D12_SHADING_RATE_COMBINER	Defines constants that specify a shading rate combiner (for variable-rate shading, or VRS).
D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER	Defines constants that specify a cross-API sharing support tier.
D3D12_SRV_DIMENSION	Identifies the type of resource that will be viewed as a shader resource.
D3D12_STATE_OBJECT_FLAGS	Specifies constraints for state objects. Use values from this enumeration in the D3D12_STATE_OBJECT_CONFIG structure.

[D3D12_STATE_OBJECT_TYPE](#)

Specifies the type of a state object. Use with D3D12_STATE_OBJECT_DESC.

[D3D12_STATE_SUBOBJECT_TYPE](#)

The type of a state subobject. Use with D3D12_STATE_SUBOBJECT.

[D3D12_STATIC_BORDER_COLOR](#)

Specifies the border color for a static sampler.

[D3D12_STENCIL_OP](#)

Identifies the stencil operations that can be performed during depth-stencil testing.

[D3D12_TEXTURE_ADDRESS_MODE](#)

Identifies a technique for resolving texture coordinates that are outside of the boundaries of a texture.

[D3D12_TEXTURE_BARRIER_FLAGS](#)

[D3D12_TEXTURE_COPY_TYPE](#)

Specifies what type of texture copy is to take place.

[D3D12_TEXTURE_LAYOUT](#)

Specifies texture layout options. (D3D12_TEXTURE_LAYOUT)

[D3D12_TILE_COPY_FLAGS](#)

Specifies how to copy a tile.

[D3D12_TILE_MAPPING_FLAGS](#)

Specifies how to perform a tile-mapping operation.

[D3D12_TILE_RANGE_FLAGS](#)

Specifies a range of tile mappings.

[D3D12_TILED_RESOURCES_TIER](#)

Identifies the tier level at which tiled resources are supported.

[D3D12_TRI_STATE](#)

TBD

[D3D12_UAV_DIMENSION](#)

Identifies unordered-access view options.

[D3D12_VARIABLE_SHADING_RATE_TIER](#)

Defines constants that specify a shading rate tier (for variable-rate shading, or VRS).

[D3D12_VIEW_INSTANCING_FLAGS](#)

Specifies options for view instancing.

[D3D12_VIEW_INSTANCING_TIER](#)

Indicates the tier level at which view instancing is supported.

[D3D12_WAVE_MMA_TIER](#)

Defines constants that specify a level of support for WaveMMA (wave_matrix) operations.

[D3D12_WRITEBUFFERIMMEDIATE_MODE](#)

Specifies the mode used by a WriteBufferImmediate operation.

Feedback

Was this page helpful?

 Yes

 No

D3D_ROOT_SIGNATURE_VERSION enumeration (d3d12.h)

Article02/14/2023

Specifies the version of root signature layout.

Syntax

C++

```
typedef enum D3D_ROOT_SIGNATURE_VERSION {
    D3D_ROOT_SIGNATURE_VERSION_1 = 0x1,
    D3D_ROOT_SIGNATURE_VERSION_1_0 = 0x1,
    D3D_ROOT_SIGNATURE_VERSION_1_1 = 0x2,
    D3D_ROOT_SIGNATURE_VERSION_1_2
};
```

Constants

[] Expand table

<code>D3D_ROOT_SIGNATURE_VERSION_1</code>
Value: <i>0x1</i>
Version one of root signature layout.

<code>D3D_ROOT_SIGNATURE_VERSION_1_0</code>
Value: <i>0x1</i>
Version one of root signature layout.

<code>D3D_ROOT_SIGNATURE_VERSION_1_1</code>
Value: <i>0x2</i>
Version 1.1 of root signature layout. Refer to Root Signature Version 1.1 .

Remarks

This enum is used by the following structures and methods.

- [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#)
- [D3D12_FEATURE_DATA_ROOT_SIGNATURE](#)

- [GetRootSignatureDescAtVersion](#)
- [D3D12SerializeRootSignature](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D_SHADER_MODEL enumeration (d3d12.h)

Article 02/22/2024

Specifies a shader model.

Syntax

C++

```
typedef enum D3D_SHADER_MODEL {
    D3D_SHADER_MODEL_NONE,
    D3D_SHADER_MODEL_5_1 = 0x51,
    D3D_SHADER_MODEL_6_0 = 0x60,
    D3D_SHADER_MODEL_6_1 = 0x61,
    D3D_SHADER_MODEL_6_2 = 0x62,
    D3D_SHADER_MODEL_6_3 = 0x63,
    D3D_SHADER_MODEL_6_4 = 0x64,
    D3D_SHADER_MODEL_6_5 = 0x65,
    D3D_SHADER_MODEL_6_6 = 0x66,
    D3D_SHADER_MODEL_6_7 = 0x67,
    D3D_SHADER_MODEL_6_8,
    D3D_SHADER_MODEL_6_9,
    D3D_HIGHEST_SHADER_MODEL
} ;
```

Constants

[] Expand table

D3D_SHADER_MODEL_5_1

Value: *0x51*

Indicates shader model 5.1.

D3D_SHADER_MODEL_6_0

Value: *0x60*

Indicates shader model 6.0. Compiling a shader model 6.0 shader requires using the DXC compiler (see [DirectX Shader Compiler](#)), and is not supported by legacy FXC.

D3D_SHADER_MODEL_6_1

Value: *0x61*

Indicates shader model 6.1.

`D3D_SHADER_MODEL_6_2`

Value: `0x62`

`D3D_SHADER_MODEL_6_3`

Value: `0x63`

`D3D_SHADER_MODEL_6_4`

Value: `0x64`

Shader model 6.4 support was added in Windows 10, Version 1903, and is required for DirectX Raytracing (DXR).

`D3D_SHADER_MODEL_6_5`

Value: `0x65`

Shader model 6.5 support was added in Windows 10, Version 2004, and is required for Direct Machine Learning.

`D3D_SHADER_MODEL_6_6`

Value: `0x66`

Shader model 6.6 support was added in Windows 11 and the DirectX 12 Agility SDK.

`D3D_SHADER_MODEL_6_7`

Value: `0x67`

Shader model 6.7 support was added in the DirectX 12 Agility SDK v1.6. See [Agility SDK 1.606.3: Shader Model 6.7 is now publicly available!](#) on the DirectX developer blog.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_SHADER_MODEL](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_AUTO_BREADCRUMB_NODE structure (d3d12.h)

Article 10/23/2023

Represents Device Removed Extended Data (DRED) auto-breadcrumb data as a node in a linked list. Each **D3D12_AUTO_BREADCRUMB_NODE** object is singly linked to the next via its `pNext` member; except for the last node in the list, which has its `pNext` set to `nullptr`.

The Direct3D 12 runtime creates one of these for each graphics command list, and tracks them in the command allocator associated with the list. When a command list is executed, the command queue information is set. After device removal is detected, the Direct3D 12 runtime links together the auto-breadcrumb nodes for any GPU work that is still outstanding.

Syntax

C++

```
typedef struct D3D12_AUTO_BREADCRUMB_NODE {
    const char                                *pCommandListDebugNameA;
    const wchar_t                             *pCommandListDebugNameW;
    const char                                *pCommandQueueDebugNameA;
    const wchar_t                             *pCommandQueueDebugNameW;
    ID3D12GraphicsCommandList                 *pCommandList;
    ID3D12CommandQueue                        *pCommandQueue;
    UINT32                                     BreadcrumbCount;
    const UINT32                               *pLastBreadcrumbValue;
    const D3D12_AUTO_BREADCRUMB_OP             *pCommandHistory;
    const D3D12_AUTO_BREADCRUMB_NODE          *pNext;
    struct                                     D3D12_AUTO_BREADCRUMB_NODE;
} D3D12_AUTO_BREADCRUMB_NODE;
```

Members

`pCommandListDebugNameA`

A pointer to the ANSI debug name of the outstanding command list (if any).

`pCommandListDebugNameW`

A pointer to the wide debug name of the outstanding command list (if any).

`pCommandQueueDebugNameA`

A pointer to the ANSI debug name of the outstanding command queue (if any).

`pCommandQueueDebugNameW`

A pointer to the wide debug name of the outstanding command queue (if any).

`pCommandList`

A pointer to the [ID3D12GraphicsCommandList interface](#) representing the outstanding command list at the time of execution.

`pCommandQueue`

A pointer to the [ID3D12CommandQueue interface](#) representing the outstanding command queue.

`BreadcrumbCount`

A `UINT32` containing the count of `D3D12_AUTO_BREADCRUMB_OP` values in the array pointed to by `pCommandHistory`.

`pLastBreadcrumbValue`

A pointer to a constant `UINT32` containing the number of `pCommandHistory` breadcrumbs ops completed. As such, the last successfully completed breadcrumb op is at index `(*pLastBreadcrumbValue - 1)` in `pCommandHistory`.

`pCommandHistory`

A pointer to a constant array of `D3D12_AUTO_BREADCRUMB_OP` values representing all of the render/compute operations recorded into the associated command list.

`pNext`

A pointer to a constant `D3D12_AUTO_BREADCRUMB_NODE` representing the next auto-breadcrumb node in the list, or `nullptr` if this is the last node.

`D3D12_AUTO_BREADCRUMB_NODE`

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_AUTO_BREADCRUMB_NODE1 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_AUTO_BREADCRUMB_NODE1 {
    const char                                *pCommandListDebugNameA;
    const wchar_t                               *pCommandListDebugNameW;
    const char                                *pCommandQueueDebugNameA;
    const wchar_t                               *pCommandQueueDebugNameW;
    ID3D12GraphicsCommandList                 *pCommandList;
    ID3D12CommandQueue                        *pCommandQueue;
    UINT                                     BreadcrumbCount;
    const UINT                                 *pLastBreadcrumbValue;
    const D3D12_AUTO_BREADCRUMB_OP             *pCommandHistory;
    const D3D12_AUTO_BREADCRUMB_NODE1          *pNext;
    struct
    {
        UINT                                     BreadcrumbContextsCount;
        D3D12_DRED_BREADCRUMB_CONTEXT           *pBreadcrumbContexts;
    } D3D12_AUTO_BREADCRUMB_NODE1;
```

Members

pCommandListDebugNameA

pCommandListDebugNameW

pCommandQueueDebugNameA

pCommandQueueDebugNameW

pCommandList

pCommandQueue

BreadcrumbCount

pLastBreadcrumbValue

pCommandHistory

pNext

D3D12_AUTO_BREADCRUMB_NODE1

BreadcrumbContextsCount

pBreadcrumbContexts

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_AUTO_BREADCRUMB_OP enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify render/compute GPU operations.

Syntax

C++

```
typedef enum D3D12_AUTO_BREADCRUMB_OP {
    D3D12_AUTO_BREADCRUMB_OP_SETMARKER,
    D3D12_AUTO_BREADCRUMB_OP_BEGINEVENT,
    D3D12_AUTO_BREADCRUMB_OP_ENDEVENT,
    D3D12_AUTO_BREADCRUMB_OP_DRAWINSTANCED,
    D3D12_AUTO_BREADCRUMB_OP_DRAWINDEXEDINSTANCED,
    D3D12_AUTO_BREADCRUMB_OP_EXECUTEINDIRECT,
    D3D12_AUTO_BREADCRUMB_OP_DISPATCH,
    D3D12_AUTO_BREADCRUMB_OP_COPYBUFFERREGION,
    D3D12_AUTO_BREADCRUMB_OP_COPYTEXTUREREGION,
    D3D12_AUTO_BREADCRUMB_OP_COPYRESOURCE,
    D3D12_AUTO_BREADCRUMB_OP_COPYTILES,
    D3D12_AUTO_BREADCRUMB_OP_RESOLVESUBRESOURCE,
    D3D12_AUTO_BREADCRUMB_OP_CLEARRENDERTARGETVIEW,
    D3D12_AUTO_BREADCRUMB_OP_CLEARUNORDEREDACCESSVIEW,
    D3D12_AUTO_BREADCRUMB_OP_CLEARDEPTHSTENCILVIEW,
    D3D12_AUTO_BREADCRUMB_OP_RESOURCEBARRIER,
    D3D12_AUTO_BREADCRUMB_OP_EXECUTEINCLUDE,
    D3D12_AUTO_BREADCRUMB_OP_PRESENT,
    D3D12_AUTO_BREADCRUMB_OP_RESOLVEQUERYDATA,
    D3D12_AUTO_BREADCRUMB_OP_BEGINSUBMISSION,
    D3D12_AUTO_BREADCRUMB_OP_ENDSUBMISSION,
    D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME,
    D3D12_AUTO_BREADCRUMB_OP_PROCESSFRAMES,
    D3D12_AUTO_BREADCRUMB_OP_ATOMICCOPYBUFFERUINT,
    D3D12_AUTO_BREADCRUMB_OP_ATOMICCOPYBUFFERUINT64,
    D3D12_AUTO_BREADCRUMB_OP_RESOLVESUBRESOURCEREGION,
    D3D12_AUTO_BREADCRUMB_OP_WRITEBUFFERIMMEDIATE,
    D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME1,
    D3D12_AUTO_BREADCRUMB_OP_SETPROTECTEDRESOURCESESSION,
    D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME2,
    D3D12_AUTO_BREADCRUMB_OP_PROCESSFRAMES1,
    D3D12_AUTO_BREADCRUMB_OP_BUILDRAYTRACINGACCELERATIONSTRUCTURE,
    D3D12_AUTO_BREADCRUMB_OP_EMITRAYTRACINGACCELERATIONSTRUCTUREPOSTBUILDINFO,
    D3D12_AUTO_BREADCRUMB_OP_COPYRAYTRACINGACCELERATIONSTRUCTURE,
    D3D12_AUTO_BREADCRUMB_OP_DISPATCHRAYS,
    D3D12_AUTO_BREADCRUMB_OP_INITIALIZEMETACOMMAND,
    D3D12_AUTO_BREADCRUMB_OP_EXECUTEMETACOMMAND,
    D3D12_AUTO_BREADCRUMB_OP_ESTIMATEMOTION,
```

```
D3D12_AUTO_BREADCRUMB_OP_RESOLVEMOTIONVECTORHEAP,  
D3D12_AUTO_BREADCRUMB_OP_SETPIPELINESTATE1,  
D3D12_AUTO_BREADCRUMB_OP_INITIALIZEEXTENSIONCOMMAND,  
D3D12_AUTO_BREADCRUMB_OP_EXECUTEEXTENSIONCOMMAND,  
D3D12_AUTO_BREADCRUMB_OP_DISPATCHMESH,  
D3D12_AUTO_BREADCRUMB_OP_ENCODEFRAME,  
D3D12_AUTO_BREADCRUMB_OP_RESOLVEENCODEROUTPUTMETADATA,  
D3D12_AUTO_BREADCRUMB_OP_BARRIER,  
D3D12_AUTO_BREADCRUMB_OP_BEGIN_COMMAND_LIST,  
D3D12_AUTO_BREADCRUMB_OP_DISPATCHGRAPH,  
D3D12_AUTO_BREADCRUMB_OP_SETPROGRAM  
} ;
```

Constants

[Expand table](#)

<code>D3D12_AUTO_BREADCRUMB_OP_SETMARKER</code> Value: (0)
<code>D3D12_AUTO_BREADCRUMB_OP_BEGINEVENT</code> Value: (1)
<code>D3D12_AUTO_BREADCRUMB_OP_ENDEVENT</code> Value: (2)
<code>D3D12_AUTO_BREADCRUMB_OP_DRAWINSTANCED</code> Value: (3)
<code>D3D12_AUTO_BREADCRUMB_OP_DRAWINDEXEDINSTANCED</code> Value: (4)
<code>D3D12_AUTO_BREADCRUMB_OP_EXECUTEINDIRECT</code> Value: (5)
<code>D3D12_AUTO_BREADCRUMB_OP_DISPATCH</code> Value: (6)
<code>D3D12_AUTO_BREADCRUMB_OP_COPYBUFFERREGION</code> Value: (7)
<code>D3D12_AUTO_BREADCRUMB_OP_COPYTEXTUREREGION</code> Value: (8)
<code>D3D12_AUTO_BREADCRUMB_OP_COPYRESOURCE</code> Value: (9)

D3D12_AUTO_BREADCRUMB_OP_COPYTILES

Value: (10)

D3D12_AUTO_BREADCRUMB_OP_RESOLVESUBRESOURCE

Value: (11)

D3D12_AUTO_BREADCRUMB_OP_CLEARRENDERTARGETVIEW

Value: (12)

D3D12_AUTO_BREADCRUMB_OP_CLEARUNORDEREDACCESSVIEW

Value: (13)

D3D12_AUTO_BREADCRUMB_OP_CLEARDEPTHSTENCILVIEW

Value: (14)

D3D12_AUTO_BREADCRUMB_OP_RESOURCEBARRIER

Value: (15)

D3D12_AUTO_BREADCRUMB_OP_EXECUTEINCLUDE

Value: (16)

D3D12_AUTO_BREADCRUMB_OP_PRESENT

Value: (17)

D3D12_AUTO_BREADCRUMB_OP_RESOLVEQUERYDATA

Value: (18)

D3D12_AUTO_BREADCRUMB_OP_BEGINSUBMISSION

Value: (19)

D3D12_AUTO_BREADCRUMB_OP_ENDSUBMISSION

Value: (20)

D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME

Value: (21)

D3D12_AUTO_BREADCRUMB_OP_PROCESSFRAMES

Value: (22)

D3D12_AUTO_BREADCRUMB_OP_ATOMICCOPYBUFFERUINT

Value: (23)

D3D12_AUTO_BREADCRUMB_OP_ATOMICCOPYBUFFERUINT64

Value: (24)

D3D12_AUTO_BREADCRUMB_OP_RESOLVESUBRESOURCEREGION

Value: (25)

D3D12_AUTO_BREADCRUMB_OP_WRITEBUFFERIMMEDIATE

Value: (26)

D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME1

Value: (27)

D3D12_AUTO_BREADCRUMB_OP_SETPROTECTEDRESOURCESESSION

Value: (28)

D3D12_AUTO_BREADCRUMB_OP_DECODEFRAME2

Value: (29)

D3D12_AUTO_BREADCRUMB_OP_PROCESSFRAMES1

Value: (30)

D3D12_AUTO_BREADCRUMB_OP_BUILDRAYTRACINGACCELERATIONSTRUCTURE

Value: (31)

D3D12_AUTO_BREADCRUMB_OP_EMITRAYTRACINGACCELERATIONSTRUCTUREPOSTBUILDINFO

Value: (32)

D3D12_AUTO_BREADCRUMB_OP_COPYRAYTRACINGACCELERATIONSTRUCTURE

Value: (33)

D3D12_AUTO_BREADCRUMB_OP_DISPATCHRAYS

Value: (34)

D3D12_AUTO_BREADCRUMB_OP_INITIALIZEMETACOMMAND

Value: (35)

D3D12_AUTO_BREADCRUMB_OP_EXECUTEMETACOMMAND

Value: (36)

D3D12_AUTO_BREADCRUMB_OP_ESTIMATEMOTION

Value: (37)

D3D12_AUTO_BREADCRUMB_OP_RESOLVEMOTIONVECTORHEAP

Value: (38)

D3D12_AUTO_BREADCRUMB_OP_SETPIPELINESTATE1

Value: (39)

D3D12_AUTO_BREADCRUMB_OP_INITIALIZEEXTENSIONCOMMAND

Value: (40)

D3D12_AUTO_BREADCRUMB_OP_EXECUTEEXTENSIONCOMMAND

Value: (41)

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core enumerations](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

[!\[\]\(935d86dfc3cfcfdf8444440b84462761_img.jpg\) Yes](#)[!\[\]\(aec3afc910ab04c559f60a6da8929d95_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_AXIS_SHADING_RATE enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify the shading rate (for variable-rate shading, or VRS) along a horizontal or vertical axis. For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

C++

```
typedef enum D3D12_AXIS_SHADING_RATE {
    D3D12_AXIS_SHADING_RATE_1X = 0,
    D3D12_AXIS_SHADING_RATE_2X = 0x1,
    D3D12_AXIS_SHADING_RATE_4X = 0x2
};
```

Constants

[+] Expand table

`D3D12_AXIS_SHADING_RATE_1X`

Value: *0*

Specifies a 1x shading rate for the axis.

`D3D12_AXIS_SHADING_RATE_2X`

Value: *0x1*

Specifies a 2x shading rate for the axis.

`D3D12_AXIS_SHADING_RATE_4X`

Value: *0x2*

Specifies a 4x shading rate for the axis.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BACKGROUND_PROCESSING_MODE enumeration (d3d12.h)

Article01/31/2022

Defines constants that specify a level of dynamic optimization to apply to GPU work that's subsequently submitted.

Syntax

C++

```
typedef enum D3D12_BACKGROUND_PROCESSING_MODE {
    D3D12_BACKGROUND_PROCESSING_MODE_ALLOWED = 0,
    D3D12_BACKGROUND_PROCESSING_MODE_ALLOW_INTRUSIVE_MEASUREMENTS,
    D3D12_BACKGROUND_PROCESSING_MODE_DISABLE_BACKGROUND_WORK,
    D3D12_BACKGROUND_PROCESSING_MODE_DISABLE_PROFILING_BY_SYSTEM
};
```

Constants

[+] Expand table

D3D12_BACKGROUND_PROCESSING_MODE_ALLOWED

Value: 0

The default setting. Specifies that the driver may instrument workloads, and dynamically recompile shaders, in a low overhead, non-intrusive manner that avoids glitching the foreground workload.

D3D12_BACKGROUND_PROCESSING_MODE_ALLOW_INTRUSIVE_MEASUREMENTS

Specifies that the driver may instrument as aggressively as possible. The understanding is that causing glitches is fine while in this mode, because the current work is being submitted specifically to train the system.

D3D12_BACKGROUND_PROCESSING_MODE_DISABLE_BACKGROUND_WORK

Specifies that background work should stop. This ensures that background shader recompilation won't consume CPU cycles. Available only in **Developer mode**.

D3D12_BACKGROUND_PROCESSING_MODE_DISABLE_PROFILING_BY_SYSTEM

Specifies that all dynamic optimization should be disabled. For example, if you're doing an A/B performance comparison, then using this constant ensures that the driver doesn't change anything that might interfere with your results. Available only in **Developer mode**.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BARRIER_ACCESS enumeration (d3d12.h)

Article11/10/2023

Syntax

C++

```
typedef enum D3D12_BARRIER_ACCESS {
    D3D12_BARRIER_ACCESS_COMMON,
    D3D12_BARRIER_ACCESS_VERTEX_BUFFER,
    D3D12_BARRIER_ACCESS_CONSTANT_BUFFER,
    D3D12_BARRIER_ACCESS_INDEX_BUFFER,
    D3D12_BARRIER_ACCESS_RENDER_TARGET,
    D3D12_BARRIER_ACCESS_UNORDERED_ACCESS,
    D3D12_BARRIER_ACCESS_DEPTH_STENCIL_WRITE,
    D3D12_BARRIER_ACCESS_DEPTH_STENCIL_READ,
    D3D12_BARRIER_ACCESS_SHADER_RESOURCE,
    D3D12_BARRIER_ACCESS_STREAM_OUTPUT,
    D3D12_BARRIER_ACCESS_INDIRECT_ARGUMENT,
    D3D12_BARRIER_ACCESS_PREDICATION,
    D3D12_BARRIER_ACCESS_COPY_DEST,
    D3D12_BARRIER_ACCESS_COPY_SOURCE,
    D3D12_BARRIER_ACCESS_RESOLVE_DEST,
    D3D12_BARRIER_ACCESS_RESOLVE_SOURCE,
    D3D12_BARRIER_ACCESS_RAYTRACING_ACCELERATION_STRUCTURE_READ,
    D3D12_BARRIER_ACCESS_RAYTRACING_ACCELERATION_STRUCTURE_WRITE,
    D3D12_BARRIER_ACCESS_SHADING_RATE_SOURCE,
    D3D12_BARRIER_ACCESS_VIDEO_DECODE_READ,
    D3D12_BARRIER_ACCESS_VIDEO_DECODE_WRITE,
    D3D12_BARRIER_ACCESS_VIDEO_PROCESS_READ,
    D3D12_BARRIER_ACCESS_VIDEO_PROCESS_WRITE,
    D3D12_BARRIER_ACCESS_VIDEO_ENCODE_READ,
    D3D12_BARRIER_ACCESS_VIDEO_ENCODE_WRITE,
    D3D12_BARRIER_ACCESS_NO_ACCESS
} ;
```

Constants

D3D12_BARRIER_ACCESS_COMMON

D3D12_BARRIER_ACCESS_VERTEX_BUFFER

D3D12_BARRIER_ACCESS_CONSTANT_BUFFER
D3D12_BARRIER_ACCESS_INDEX_BUFFER
D3D12_BARRIER_ACCESS_RENDER_TARGET
D3D12_BARRIER_ACCESS_UNORDERED_ACCESS
D3D12_BARRIER_ACCESS_DEPTH_STENCIL_WRITE
D3D12_BARRIER_ACCESS_DEPTH_STENCIL_READ
D3D12_BARRIER_ACCESS_SHADER_RESOURCE
D3D12_BARRIER_ACCESS_STREAM_OUTPUT
D3D12_BARRIER_ACCESS_INDIRECT_ARGUMENT
D3D12_BARRIER_ACCESS_PREDICATION
D3D12_BARRIER_ACCESS_COPY_DEST
D3D12_BARRIER_ACCESS_COPY_SOURCE
D3D12_BARRIER_ACCESS_RESOLVE_DEST
D3D12_BARRIER_ACCESS_RESOLVE_SOURCE
D3D12_BARRIER_ACCESS_RAYTRACING_ACCELERATION_STRUCTURE_READ
D3D12_BARRIER_ACCESS_RAYTRACING_ACCELERATION_STRUCTURE_WRITE
D3D12_BARRIER_ACCESS_SHADING_RATE_SOURCE
D3D12_BARRIER_ACCESS_VIDEO_DECODE_READ
D3D12_BARRIER_ACCESS_VIDEO_DECODE_WRITE
D3D12_BARRIER_ACCESS_VIDEO_PROCESS_READ
D3D12_BARRIER_ACCESS_VIDEO_PROCESS_WRITE
D3D12_BARRIER_ACCESS_VIDEO_ENCODE_READ
D3D12_BARRIER_ACCESS_VIDEO_ENCODE_WRITE
D3D12_BARRIER_ACCESS_NO_ACCESS

Requirements

Header

d3d12.h

Feedback

Was this page helpful?

 Yes

 No

D3D12_BARRIER_GROUP structure (d3d12.h)

Article 02/22/2024

Describes a group of barriers of a given type.

Syntax

C++

```
typedef struct D3D12_BARRIER_GROUP {
    D3D12_BARRIER_TYPE Type;
    UINT32             NumBarriers;
    union {
        const D3D12_GLOBAL_BARRIER *pGlobalBarriers;
        const D3D12_TEXTURE_BARRIER *pTextureBarriers;
        const D3D12_BUFFER_BARRIER *pBufferBarriers;
    };
} D3D12_BARRIER_GROUP;
```

Members

Type

The type of barriers in the group.

NumBarriers

The number of barriers in the group.

pGlobalBarriers

A pointer to an array of [D3D12_GLOBAL_BARRIER](#) structures, if *Type* is `D3D12_BARRIER_TYPE::D3D12_BARRIER_TYPE_GLOBAL`.

pTextureBarriers

A pointer to an array of [D3D12_TEXTURE_BARRIER](#) structures, if *Type* is `D3D12_BARRIER_TYPE::D3D12_BARRIER_TYPE_TEXTURE`.

pBufferBarriers

A pointer to an array of [D3D12_BUFFER_BARRIER](#) structures, if *Type* is `D3D12_BARRIER_TYPE::D3D12_BARRIER_TYPE_BUFFER`.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BARRIER_LAYOUT enumeration (d3d12.h)

Article11/10/2023

Syntax

C++

```
typedef enum D3D12_BARRIER_LAYOUT {
    D3D12_BARRIER_LAYOUT_UNDEFINED,
    D3D12_BARRIER_LAYOUT_COMMON,
    D3D12_BARRIER_LAYOUT_PRESENT,
    D3D12_BARRIER_LAYOUT_GENERIC_READ,
    D3D12_BARRIER_LAYOUT_RENDER_TARGET,
    D3D12_BARRIER_LAYOUT_UNORDERED_ACCESS,
    D3D12_BARRIER_LAYOUT_DEPTH_STENCIL_WRITE,
    D3D12_BARRIER_LAYOUT_DEPTH_STENCIL_READ,
    D3D12_BARRIER_LAYOUT_SHADER_RESOURCE,
    D3D12_BARRIER_LAYOUT_COPY_SOURCE,
    D3D12_BARRIER_LAYOUT_COPY_DEST,
    D3D12_BARRIER_LAYOUT_RESOLVE_SOURCE,
    D3D12_BARRIER_LAYOUT_RESOLVE_DEST,
    D3D12_BARRIER_LAYOUT_SHADING_RATE_SOURCE,
    D3D12_BARRIER_LAYOUT_VIDEO_DECODE_READ,
    D3D12_BARRIER_LAYOUT_VIDEO_DECODE_WRITE,
    D3D12_BARRIER_LAYOUT_VIDEO_PROCESS_READ,
    D3D12_BARRIER_LAYOUT_VIDEO_PROCESS_WRITE,
    D3D12_BARRIER_LAYOUT_VIDEO_ENCODE_READ,
    D3D12_BARRIER_LAYOUT_VIDEO_ENCODE_WRITE,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COMMON,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_GENERIC_READ,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_UNORDERED_ACCESS,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_SHADER_RESOURCE,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COPY_SOURCE,
    D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COPY_DEST,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COMMON,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_GENERIC_READ,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_UNORDERED_ACCESS,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_SHADER_RESOURCE,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COPY_SOURCE,
    D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COPY_DEST,
    D3D12_BARRIER_LAYOUT_VIDEO_QUEUE_COMMON
};
```

Constants

D3D12_BARRIER_LAYOUT_UNDEFINED

D3D12_BARRIER_LAYOUT_COMMON

D3D12_BARRIER_LAYOUT_PRESENT

D3D12_BARRIER_LAYOUT_GENERIC_READ

D3D12_BARRIER_LAYOUT_RENDER_TARGET

D3D12_BARRIER_LAYOUT_UNORDERED_ACCESS

D3D12_BARRIER_LAYOUT_DEPTH_STENCIL_WRITE

D3D12_BARRIER_LAYOUT_DEPTH_STENCIL_READ

D3D12_BARRIER_LAYOUT_SHADER_RESOURCE

D3D12_BARRIER_LAYOUT_COPY_SOURCE

D3D12_BARRIER_LAYOUT_COPY_DEST

D3D12_BARRIER_LAYOUT_RESOLVE_SOURCE

D3D12_BARRIER_LAYOUT_RESOLVE_DEST

D3D12_BARRIER_LAYOUT_SHADING_RATE_SOURCE

D3D12_BARRIER_LAYOUT_VIDEO_DECODE_READ

D3D12_BARRIER_LAYOUT_VIDEO_DECODE_WRITE

D3D12_BARRIER_LAYOUT_VIDEO_PROCESS_READ

D3D12_BARRIER_LAYOUT_VIDEO_PROCESS_WRITE

D3D12_BARRIER_LAYOUT_VIDEO_ENCODE_READ

D3D12_BARRIER_LAYOUT_VIDEO_ENCODE_WRITE

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COMMON

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_GENERIC_READ

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_UNORDERED_ACCESS

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_SHADER_RESOURCE

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COPY_SOURCE

D3D12_BARRIER_LAYOUT_DIRECT_QUEUE_COPY_DEST

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COMMON

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_GENERIC_READ

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_UNORDERED_ACCESS

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_SHADER_RESOURCE

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COPY_SOURCE

D3D12_BARRIER_LAYOUT_COMPUTE_QUEUE_COPY_DEST

D3D12_BARRIER_LAYOUT_VIDEO_QUEUE_COMMON

Requirements

Header	d3d12.h
--------	---------

Feedback

Was this page helpful?

 Yes

 No

D3D12_BARRIER_SUBRESOURCE_RANGE structure (d3d12.h)

Article02/22/2024

Allows you to transition logically-adjacent ranges of subresources.

Syntax

C++

```
typedef struct D3D12_BARRIER_SUBRESOURCE_RANGE {
    UINT IndexOrFirstMipLevel;
    UINT NumMipLevels;
    UINT FirstArraySlice;
    UINT NumArraySlices;
    UINT FirstPlane;
    UINT NumPlanes;
} D3D12_BARRIER_SUBRESOURCE_RANGE;
```

Members

`IndexOrFirstMipLevel`

The index of the first mip level in the range; or a subresource index, if *NumMipLevels* is zero. If a subresource index, then you can use the value `0xffffffff` to specify all subresources.

`NumMipLevels`

Number of mip levels in the range, or zero to indicate that *IndexOrFirstMipLevel* is a subresource index.

`FirstArraySlice`

Index of first array slice in the range. Ignored if *NumMipLevels* is zero.

`NumArraySlices`

Number of array slices in the range. Ignored if *NumMipLevels* is zero.

`FirstPlane`

First plane slice in the range. Ignored if *NumMipLevels* is zero.

NumPlanes

Number of plane slices in the range. Ignored if *NumMipLevels* is zero.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_BARRIER_SYNC enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_BARRIER_SYNC {
    D3D12_BARRIER_SYNC_NONE,
    D3D12_BARRIER_SYNC_ALL,
    D3D12_BARRIER_SYNC_DRAW,
    D3D12_BARRIER_SYNC_INDEX_INPUT,
    D3D12_BARRIER_SYNC_VERTEX_SHADING,
    D3D12_BARRIER_SYNC_PIXEL_SHADING,
    D3D12_BARRIER_SYNC_DEPTH_STENCIL,
    D3D12_BARRIER_SYNC_RENDER_TARGET,
    D3D12_BARRIER_SYNC_COMPUTE_SHADING,
    D3D12_BARRIER_SYNC_RAYTRACING,
    D3D12_BARRIER_SYNC_COPY,
    D3D12_BARRIER_SYNC_RESOLVE,
    D3D12_BARRIER_SYNC_EXECUTE_INDIRECT,
    D3D12_BARRIER_SYNC_PREDICATION,
    D3D12_BARRIER_SYNC_ALL_SHADING,
    D3D12_BARRIER_SYNC_NON_PIXEL_SHADING,
    D3D12_BARRIER_SYNC_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO,
    D3D12_BARRIER_SYNC_CLEAR_UNORDERED_ACCESS_VIEW,
    D3D12_BARRIER_SYNC_VIDEO_DECODE,
    D3D12_BARRIER_SYNC_VIDEO_PROCESS,
    D3D12_BARRIER_SYNC_VIDEO_ENCODE,
    D3D12_BARRIER_SYNC_BUILD_RAYTRACING_ACCELERATION_STRUCTURE,
    D3D12_BARRIER_SYNC_COPY_RAYTRACING_ACCELERATION_STRUCTURE,
    D3D12_BARRIER_SYNC_SPLIT
};
```

Constants

[+] Expand table

D3D12_BARRIER_SYNC_NONE

D3D12_BARRIER_SYNC_ALL

D3D12_BARRIER_SYNC_DRAW
D3D12_BARRIER_SYNC_VERTEX_SHADING
D3D12_BARRIER_SYNC_PIXEL_SHADING
D3D12_BARRIER_SYNC_DEPTH_STENCIL
D3D12_BARRIER_SYNC_RENDER_TARGET
D3D12_BARRIER_SYNC_COMPUTE_SHADING
D3D12_BARRIER_SYNC_RAYTRACING
D3D12_BARRIER_SYNC_COPY
D3D12_BARRIER_SYNC_RESOLVE
D3D12_BARRIER_SYNC_EXECUTE_INDIRECT
D3D12_BARRIER_SYNC_PREDICATION
D3D12_BARRIER_SYNC_ALL_SHADING
D3D12_BARRIER_SYNC_NON_PIXEL_SHADING
D3D12_BARRIER_SYNC_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO
D3D12_BARRIER_SYNC_VIDEO_DECODE
D3D12_BARRIER_SYNC_VIDEO_PROCESS
D3D12_BARRIER_SYNC_VIDEO_ENCODE
D3D12_BARRIER_SYNC_BUILD_RAYTRACING_ACCELERATION_STRUCTURE
D3D12_BARRIER_SYNC_COPY_RAYTRACING_ACCELERATION_STRUCTURE
D3D12_BARRIER_SYNC_SPLIT

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BARRIER_TYPE enumeration (d3d12.h)

Article 02/22/2024

Syntax

C++

```
typedef enum D3D12_BARRIER_TYPE {
    D3D12_BARRIER_TYPE_GLOBAL,
    D3D12_BARRIER_TYPE_TEXTURE,
    D3D12_BARRIER_TYPE_BUFFER
};
```

Constants

 Expand table

D3D12_BARRIER_TYPE_GLOBAL
D3D12_BARRIER_TYPE_TEXTURE
D3D12_BARRIER_TYPE_BUFFER

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

D3D12_BLEND enumeration (d3d12.h)

Article11/03/2022

Specifies blend factors, which modulate values for the pixel shader and render target.

Syntax

C++

```
typedef enum D3D12_BLEND {
    D3D12_BLEND_ZERO = 1,
    D3D12_BLEND_ONE = 2,
    D3D12_BLEND_SRC_COLOR = 3,
    D3D12_BLEND_INV_SRC_COLOR = 4,
    D3D12_BLEND_SRC_ALPHA = 5,
    D3D12_BLEND_INV_SRC_ALPHA = 6,
    D3D12_BLEND_DEST_ALPHA = 7,
    D3D12_BLEND_INV_DEST_ALPHA = 8,
    D3D12_BLEND_DEST_COLOR = 9,
    D3D12_BLEND_INV_DEST_COLOR = 10,
    D3D12_BLEND_SRC_ALPHA_SAT = 11,
    D3D12_BLEND_BLEND_FACTOR = 14,
    D3D12_BLEND_INV_BLEND_FACTOR = 15,
    D3D12_BLEND_SRC1_COLOR = 16,
    D3D12_BLEND_INV_SRC1_COLOR = 17,
    D3D12_BLEND_SRC1_ALPHA = 18,
    D3D12_BLEND_INV_SRC1_ALPHA = 19,
    D3D12_BLEND_ALPHA_FACTOR = 20,
    D3D12_BLEND_INV_ALPHA_FACTOR = 21
};
```

Constants

[] Expand table

D3D12_BLEND_ZERO
Value: 1
The blend factor is (0, 0, 0, 0). No pre-blend operation.

D3D12_BLEND_ONE
Value: 2
The blend factor is (1, 1, 1, 1). No pre-blend operation.

D3D12_BLEND_SRC_COLOR

Value: 3

The blend factor is (R_s, G_s, B_s, A_s) , that is color data (RGB) from a pixel shader. No pre-blend operation.

D3D12_BLEND_INV_SRC_COLOR

Value: 4

The blend factor is $(1 - R_s, 1 - G_s, 1 - B_s, 1 - A_s)$, that is color data (RGB) from a pixel shader. The pre-blend operation inverts the data, generating 1 - RGB.

D3D12_BLEND_SRC_ALPHA

Value: 5

The blend factor is (A_s, A_s, A_s, A_s) , that is alpha data (A) from a pixel shader. No pre-blend operation.

D3D12_BLEND_INV_SRC_ALPHA

Value: 6

The blend factor is $(1 - A_s, 1 - A_s, 1 - A_s, 1 - A_s)$, that is alpha data (A) from a pixel shader. The pre-blend operation inverts the data, generating 1 - A.

D3D12_BLEND_DEST_ALPHA

Value: 7

The blend factor is (A_d, A_d, A_d, A_d) , that is alpha data from a render target. No pre-blend operation.

D3D12_BLEND_INV_DEST_ALPHA

Value: 8

The blend factor is $(1 - A_d, 1 - A_d, 1 - A_d, 1 - A_d)$, that is alpha data from a render target. The pre-blend operation inverts the data, generating 1 - A.

D3D12_BLEND_DEST_COLOR

Value: 9

The blend factor is (R_d, G_d, B_d, A_d) , that is color data from a render target. No pre-blend operation.

D3D12_BLEND_INV_DEST_COLOR

Value: 10

The blend factor is $(1 - R_d, 1 - G_d, 1 - B_d, 1 - A_d)$, that is color data from a render target. The pre-blend operation inverts the data, generating 1 - RGB.

D3D12_BLEND_SRC_ALPHA_SAT

Value: 11

The blend factor is $(f, f, f, 1)$; where $f = \min(A_s, 1 - A_d)$. The pre-blend operation clamps the data to 1 or less.

D3D12_BLEND_BLEND_FACTOR

Value: 14

The blend factor is the blend factor set with [ID3D12GraphicsCommandList::OMSetBlendFactor](#). No pre-blend operation.

`D3D12_BLEND_INV_BLEND_FACTOR`

Value: 15

The blend factor is the blend factor set with [ID3D12GraphicsCommandList::OMSetBlendFactor](#).

The pre-blend operation inverts the blend factor, generating $1 - \text{blend_factor}$.

`D3D12_BLEND_SRC1_COLOR`

Value: 16

The blend factor is data sources both as color data output by a pixel shader. There is no pre-blend operation. This blend factor supports dual-source color blending.

`D3D12_BLEND_INV_SRC1_COLOR`

Value: 17

The blend factor is data sources both as color data output by a pixel shader. The pre-blend operation inverts the data, generating $1 - \text{RGB}$. This blend factor supports dual-source color blending.

`D3D12_BLEND_SRC1_ALPHA`

Value: 18

The blend factor is data sources as alpha data output by a pixel shader. There is no pre-blend operation. This blend factor supports dual-source color blending.

`D3D12_BLEND_INV_SRC1_ALPHA`

Value: 19

The blend factor is data sources as alpha data output by a pixel shader. The pre-blend operation inverts the data, generating $1 - \text{A}$. This blend factor supports dual-source color blending.

`D3D12_BLEND_ALPHA_FACTOR`

Value: 20

The blend factor is $(\text{A}, \text{A}, \text{A}, \text{A})$, where the constant, A, is taken from the blend factor set with [OMSetBlendFactor](#).

To successfully use this constant on a target machine, the

[D3D12_FEATURE_DATA_D3D12_OPTIONS13](#) returned from [capability querying](#) must have its *AlphaBlendFactorSupported* set to `TRUE`.

`D3D12_BLEND_INV_ALPHA_FACTOR`

Value: 21

The blend factor is $(1 - \text{A}, 1 - \text{A}, 1 - \text{A}, 1 - \text{A})$, where the constant, A, is taken from the blend factor set with [OMSetBlendFactor](#).

To successfully use this constant on a target machine, the

[D3D12_FEATURE_DATA_D3D12_OPTIONS13](#) returned from [capability querying](#) must have its *AlphaBlendFactorSupported* set to `TRUE`.

Remarks

Source and destination blend operations are specified in a [D3D12_RENDER_TARGET_BLEND_DESC](#) structure.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BLEND_DESC structure (d3d12.h)

Article 02/22/2024

Describes the blend state.

Syntax

C++

```
typedef struct D3D12_BLEND_DESC {
    BOOL AlphaToCoverageEnable;
    BOOL IndependentBlendEnable;
    D3D12_RENDER_TARGET_BLEND_DESC RenderTarget[8];
} D3D12_BLEND_DESC;
```

Members

`AlphaToCoverageEnable`

Specifies whether to use alpha-to-coverage as a multisampling technique when setting a pixel to a render target. For more info about using alpha-to-coverage, see [Alpha-To-Coverage](#).

`IndependentBlendEnable`

Specifies whether to enable independent blending in simultaneous render targets. Set to **TRUE** to enable independent blending. If set to **FALSE**, only the `RenderTarget[0]` members are used; `RenderTarget[1..7]` are ignored.

See the **Remarks** section for restrictions.

`RenderTarget[8]`

An array of `D3D12_RENDER_TARGET_BLEND_DESC` structures that describe the blend states for render targets; these correspond to the eight render targets that can be bound to the [output-merger stage](#) at one time.

Remarks

A `D3D12_GRAPHICS_PIPELINE_STATE_DESC` object contains a blend-state structure that controls blending by the output-merger stage.

Here are the default values for blend state.

[+] Expand table

State	Default Value
AlphaToCoverageEnable	FALSE
IndependentBlendEnable	FALSE
RenderTarget[0].BlendEnable	FALSE
RenderTarget[0].LogicOpEnable	FALSE
RenderTarget[0].SrcBlend	D3D12_BLEND_ONE
RenderTarget[0].DestBlend	D3D12_BLEND_ZERO
RenderTarget[0].BlendOp	D3D12_BLEND_OP_ADD
RenderTarget[0].SrcBlendAlpha	D3D12_BLEND_ONE
RenderTarget[0].DestBlendAlpha	D3D12_BLEND_ZERO
RenderTarget[0].BlendOpAlpha	D3D12_BLEND_OP_ADD
RenderTarget[0].LogicOp	D3D12_LOGIC_OP_NOOP
RenderTarget[0].RenderTargetWriteMask	D3D12_COLOR_WRITE_ENABLE_ALL

When you set the **LogicOpEnable** member of the first element of the **RenderTarget** array (**RenderTarget[0]**) to **TRUE**, you must also set the **BlendEnable** member of **RenderTarget[0]** to **FALSE**, and the **IndependentBlendEnable** member of this structure to **FALSE**. This reflects the limitation in hardware that you can't mix logic operations with blending across multiple render targets, and that when you use a logic operation, you must apply the same logic operation to all render targets.

Note the helper structure, [CD3DX12_BLEND_DESC](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BLEND_OP enumeration (d3d12.h)

Article01/31/2022

Specifies RGB or alpha blending operations.

Syntax

C++

```
typedef enum D3D12_BLEND_OP {
    D3D12_BLEND_OP_ADD = 1,
    D3D12_BLEND_OP_SUBTRACT = 2,
    D3D12_BLEND_OP_REV_SUBTRACT = 3,
    D3D12_BLEND_OP_MIN = 4,
    D3D12_BLEND_OP_MAX = 5
} ;
```

Constants

[] Expand table

	D3D12_BLEND_OP_ADD
Value:	1
Add source 1 and source 2.	
	D3D12_BLEND_OP_SUBTRACT
Value:	2
Subtract source 1 from source 2.	
	D3D12_BLEND_OP_REV_SUBTRACT
Value:	3
Subtract source 2 from source 1.	
	D3D12_BLEND_OP_MIN
Value:	4
Find the minimum of source 1 and source 2.	
	D3D12_BLEND_OP_MAX
Value:	5
Find the maximum of source 1 and source 2.	

Remarks

The runtime implements RGB blending and alpha blending separately. Therefore, blend state requires separate blend operations for RGB data and alpha data. These blend operations are specified in a [D3D12_RENDER_TARGET_BLEND_DESC](#) structure. The two sources—source 1 and source 2—are shown in the [blending block diagram](#).

Blend state is used by the [output-merger stage](#) to determine how to blend together two RGB pixel values and two alpha values. The two RGB pixel values and two alpha values are the RGB pixel value and alpha value that the pixel shader outputs and the RGB pixel value and alpha value already in the output render target. The [D3D12_BLEND](#) value controls the data source that the blending stage uses to modulate values for the pixel shader, render target, or both. The [D3D12_BLEND_OP](#) value controls how the blending stage mathematically combines these modulated values.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BOX structure (d3d12.h)

Article 02/22/2024

Describes a 3D box.

Syntax

C++

```
typedef struct D3D12_BOX {
    UINT left;
    UINT top;
    UINT front;
    UINT right;
    UINT bottom;
    UINT back;
} D3D12_BOX;
```

Members

`left`

The x position of the left hand side of the box.

`top`

The y position of the top of the box.

`front`

The z position of the front of the box.

`right`

The x position of the right hand side of the box, plus 1. This means that `right - left` equals the width of the box.

`bottom`

The y position of the bottom of the box, plus 1. This means that `bottom - top` equals the height of the box.

`back`

The z position of the back of the box, plus 1. This means that `back - front` equals the depth of the box.

Remarks

This structure is used by the methods [WriteToSubresource](#), [ReadFromSubresource](#) and [CopyTextureRegion](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_BOX](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_BARRIER structure (d3d12.h)

Article02/22/2024

Describes a buffer memory access barrier. Used by buffer barriers to indicate when resource memory must be made visible for a specific access type.

Syntax

C++

```
typedef struct D3D12_BUFFER_BARRIER {
    D3D12_BARRIER_SYNC SyncBefore;
    D3D12_BARRIER_SYNC SyncAfter;
    D3D12_BARRIER_ACCESS AccessBefore;
    D3D12_BARRIER_ACCESS AccessAfter;
    ID3D12Resource *pResource;
    UINT64 Offset;
    UINT64 Size;
} D3D12_BUFFER_BARRIER;
```

Members

SyncBefore

Synchronization scope of all preceding GPU work that must be completed before executing the barrier.

SyncAfter

Synchronization scope of all subsequent GPU work that must wait until the barrier execution is finished.

AccessBefore

Access bits corresponding with resource usage since the preceding barrier, or the start of **ExecuteCommandLists** scope.

AccessAfter

Access bits corresponding with resource usage after the barrier completes.

`pResource`

Pointer to the buffer resource being using the barrier.

`Offset`

Must be 0.

`Size`

Must be either `UINT64_MAX` or the size of the buffer in bytes.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_RTV structure (d3d12.h)

Article 02/22/2024

Describes the elements in a buffer resource to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_BUFFER_RTV {
    UINT64 FirstElement;
    UINT    NumElements;
} D3D12_BUFFER_RTV;
```

Members

`FirstElement`

Number of elements between the beginning of the buffer and the first element to access.

`NumElements`

The total number of elements in the view.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as a buffer.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_SRV structure (d3d12.h)

Article 02/22/2024

Describes the elements in a buffer resource to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_BUFFER_SRV {
    UINT64 FirstElement;
    UINT NumElements;
    UINT StructureByteStride;
    D3D12_BUFFER_SRV_FLAGS Flags;
} D3D12_BUFFER_SRV;
```

Members

`FirstElement`

The index of the first element to be accessed by the view.

`NumElements`

The number of elements in the resource.

`StructureByteStride`

The size of each element in the buffer structure (in bytes) when the buffer represents a structured buffer. The size must match the struct size declared in shaders that access the view.

`Flags`

A `D3D12_BUFFER_SRV_FLAGS`-typed value that identifies view options for the buffer. Currently, the only option is to identify a raw view of the buffer. For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

Remarks

This structure is used by `D3D12_SHADER_RESOURCE_VIEW_DESC` to create a view of a buffer.

If the value of *StructureByteStride* is not 0, then a view of a structured buffer is created, and then the *D3D12_SHADER_RESOURCE_VIEW_DESC::Format* field must be **DXGI_FORMAT_UNKNOWN**. If *StructureByteStride* is 0, then a typed view of a buffer is created, and then a format must be supplied. The specified format for the typed view must be supported by the hardware.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_SRV_FLAGS enumeration (d3d12.h)

Article02/22/2024

Identifies how to view a buffer resource.

Syntax

C++

```
typedef enum D3D12_BUFFER_SRV_FLAGS {
    D3D12_BUFFER_SRV_FLAG_NONE = 0,
    D3D12_BUFFER_SRV_FLAG_RAW = 0x1
} ;
```

Constants

[+] Expand table

D3D12_BUFFER_SRV_FLAG_NONE

Value: 0

Indicates a default view.

D3D12_BUFFER_SRV_FLAG_RAW

Value: 0x1

View the buffer as raw. For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

Remarks

This enumeration is used by [D3D12_BUFFER_SRV](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_UAV structure (d3d12.h)

Article 07/27/2022

Describes the elements in a buffer to use in a unordered-access view.

Syntax

C++

```
typedef struct D3D12_BUFFER_UAV {
    UINT64 FirstElement;
    UINT NumElements;
    UINT StructureByteStride;
    UINT64 CounterOffsetInBytes;
    D3D12_BUFFER_UAV_FLAGS Flags;
} D3D12_BUFFER_UAV;
```

Members

`FirstElement`

The zero-based index of the first element to be accessed.

`NumElements`

The number of elements in the resource. For structured buffers, this is the number of structures in the buffer.

`StructureByteStride`

The size of each element in the buffer structure (in bytes) when the buffer represents a structured buffer.

`CounterOffsetInBytes`

The counter offset, in bytes.

`Flags`

A [D3D12_BUFFER_UAV_FLAGS](#)-typed value that specifies the view options for the resource.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as a buffer.

If *StructureByteStride* value is not 0, a view of a structured buffer is created and the [D3D12_UNORDERED_ACCESS_VIEW_DESC::Format](#) field must be [DXGI_FORMAT_UNKNOWN](#). If *StructureByteStride* is 0, a typed view of a buffer is created and a format must be supplied. The specified format for the typed view must be supported by the hardware. More information on this topic can be found in the [Typed unordered access view \(UAV\) loads](#) page.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Typed unordered access view \(UAV\) loads](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUFFER_UAV_FLAGS enumeration (d3d12.h)

Article02/22/2024

Identifies unordered-access view options for a buffer resource.

Syntax

C++

```
typedef enum D3D12_BUFFER_UAV_FLAGS {
    D3D12_BUFFER_UAV_FLAG_NONE = 0,
    D3D12_BUFFER_UAV_FLAG_RAW = 0x1
} ;
```

Constants

[] Expand table

`D3D12_BUFFER_UAV_FLAG_NONE`

Value: *0*

Indicates a default view.

`D3D12_BUFFER_UAV_FLAG_RAW`

Value: *0x1*

Resource contains raw, unstructured data. Requires the UAV format to be [DXGI_FORMAT_R32_TYPELESS](#).

For more info about raw viewing of buffers, see [Raw Views of Buffers](#).

Remarks

This enum is used in the [D3D12_BUFFER_UAV](#) structure.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC structure (d3d12.h)

Article04/02/2021

Describes a raytracing acceleration structure. Pass this structure into [ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure](#) to describe the acceleration structure to be built.

Syntax

C++

```
typedef struct D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC {
    D3D12_GPU_VIRTUAL_ADDRESS DestAccelerationStructureData;
    D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS Inputs;
    D3D12_GPU_VIRTUAL_ADDRESS SourceAccelerationStructureData;
    D3D12_GPU_VIRTUAL_ADDRESS ScratchAccelerationStructureData;
} D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC;
```

Members

`DestAccelerationStructureData`

Location to store resulting acceleration structure.

[ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo](#) reports the amount of memory required for the result here given a set of acceleration structure build parameters.

The address must be aligned to 256 bytes, defined as [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT](#).

Important

The memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#)

Inputs

Description of the input data for the acceleration structure build. This is data is stored in a separate structure because it is also used with [GetRaytracingAccelerationStructurePrebuildInfo](#).

SourceAccelerationStructureData

Address of an existing acceleration structure if an acceleration structure update (an incremental build) is being requested, by setting [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE](#) in the Flags parameter. Otherwise this address must be NULL.

If this address is the same as *DestAccelerationStructureData*, the update is to be performed in-place. Any other form of overlap of the source and destination memory is invalid and produces undefined behavior.

The address must be aligned to 256 bytes, defined as

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT](#), which should automatically be the case because any existing acceleration structure passed in here would have already been required to be placed with such alignment.

ⓘ Important

The memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#).

ScratchAccelerationStructureData

Location where the build will store temporary data.

[GetRaytracingAccelerationStructurePrebuildInfo](#) reports the amount of scratch memory the implementation will need for a given set of acceleration structure build parameters.

ⓘ Important

The memory must be in state [D3D12 RESOURCE STATE UNORDERED ACCESS](#).

ScratchAccelerationStructureData

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS structure (d3d12.h)

Article04/02/2021

Defines the inputs for a raytracing acceleration structure build operation. This structure is used by [ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure](#) and [ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo](#).

Syntax

C++

```
typedef struct D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE          Type;
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS Flags;
    UINT                                                    NumDescs;
    D3D12_ELEMENTS_LAYOUT                                DescsLayout;
    union {
        D3D12_GPU_VIRTUAL_ADDRESS           InstanceDescs;
        const D3D12_RAYTRACING_GEOMETRY_DESC *pGeometryDescs;
        const D3D12_RAYTRACING_GEOMETRY_DESC const ** ppGeometryDescs;
    };
} D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS;
```

Members

Type

The type of acceleration structure to build.

Flags

The build flags.

NumDescs

If *Type* is **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TOP_LEVEL**, this value is the number of instances, laid out based on *DescsLayout*.

If *Type* is **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BOTTOM_LEVEL**, this value is the number of elements referred to by *pGeometryDescs* or *ppGeometryDescs*. Which of

these fields is used depends on *DescsLayout*.

DescsLayout

How geometry descriptions are specified; either an array of descriptions or an array of pointers to descriptions.

InstanceDescs

If *Type* is **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TOP_LEVEL**, this refers to *NumDescs* [D3D12_RAYTRACING_INSTANCE_DESC](#) structures in GPU memory describing instances. Each instance must be aligned to 16 bytes, defined as [D3D12_RAYTRACING_INSTANCE_DESC_BYTE_ALIGNMENT](#).

If *Type* is not **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TOP_LEVEL**, this parameter is unused.

If *DescLayout* is **D3D12_ELEMENTS_LAYOUT_ARRAY**, *InstanceDescs* points to an array of instance descriptions in GPU memory.

If *DescLayout* is **D3D12_ELEMENTS_LAYOUT_ARRAY_OF_POINTERS**, *InstanceDescs* points to an array in GPU memory of [D3D12_GPU_VIRTUAL_ADDRESS](#) pointers to instance descriptions.

The memory pointed to must be in state

[D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

pGeometryDescs

If *Type* is **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BOTTOM_LEVEL**, and *DescsLayout* is **D3D12_ELEMENTS_LAYOUT_ARRAY**, this field is used and points to *NumDescs* contiguous [D3D12_RAYTRACING_GEOMETRY_DESC](#) structures on the CPU, describing individual geometries.

If *Type* is not **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BOTTOM_LEVEL** or *DescsLayout* is not **D3D12_ELEMENTS_LAYOUT_ARRAY**, this parameter is unused.

ppGeometryDescs

If *Type* is **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BOTTOM_LEVEL**, and *DescsLayout* is **D3D12_ELEMENTS_LAYOUT_ARRAY_OF_POINTERS**, this field is used and points to an array of *NumDescs* pointers to [D3D12_RAYTRACING_GEOMETRY_DESC](#) structures on the CPU, describing individual geometries.

Remarks

When used with [GetRaytracingAccelerationStructurePrebuildInfo](#), which actually perform a build, any parameter that is referenced via [D3D12_GPU_VIRTUAL_ADDRESS](#) (an address in GPU memory), like *InstanceDescs*, will not be accessed by the operation. So this memory does not need to be initialized yet or be in a particular resource state. Whether GPU addresses are null or not can be inspected by the operation, even though the pointers are not dereferenced.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER structure (d3d12.h)

Article02/22/2024

Describes the GPU memory layout of an acceleration structure visualization.

Syntax

C++

```
typedef struct
D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE Type;
    UINT                                         NumDescs;
} D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER;
```

Members

Type

The type of acceleration structure.

NumDescs

The number of descriptions.

Remarks

This structure functions like the inverse of the inputs to an acceleration structure build, focused on the instance or geometry details, depending on the acceleration structure type.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CACHED_PIPELINE_STATE structure (d3d12.h)

Article02/22/2024

Stores a pipeline state.

Syntax

C++

```
typedef struct D3D12_CACHED_PIPELINE_STATE {
    const void *pCachedBlob;
    SIZE_T     CachedBlobSizeInBytes;
} D3D12_CACHED_PIPELINE_STATE;
```

Members

pCachedBlob

Specifies pointer that references the memory location of the cache.

CachedBlobSizeInBytes

Specifies the size of the cache in bytes.

Remarks

This structure is used by the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure, and the [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structure.

This structure is intended to be filled with the data retrieved from [ID3D12PipelineState::GetCachedBlob](#). This cached PSO contains data specific to the hardware, driver, and machine that it was retrieved from. Compilation using this data should be faster than compilation without. The rest of the data in the PSO needs to still be valid, and needs to match the cached PSO, otherwise E_INVALIDARG might be returned.

If the driver has been upgraded to a D3D12 driver after the PSO was cached, you might see a D3D12_ERROR_DRIVER_VERSION_MISMATCH return code, or if you're running on a different GPU, the D3D12_ERROR_ADAPTER_NOT_FOUND return code.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CLEAR_FLAGS enumeration (d3d12.h)

Article 02/22/2024

Specifies what to clear from the depth stencil view.

Syntax

C++

```
typedef enum D3D12_CLEAR_FLAGS {
    D3D12_CLEAR_FLAG_DEPTH = 0x1,
    D3D12_CLEAR_FLAG_STENCIL = 0x2
};
```

Constants

[] Expand table

`D3D12_CLEAR_FLAG_DEPTH`

Value: `0x1`

Indicates the depth buffer should be cleared.

`D3D12_CLEAR_FLAG_STENCIL`

Value: `0x2`

Indicates the stencil buffer should be cleared.

Remarks

This enum is used by [ID3D12GraphicsCommandList::ClearDepthStencilView](#). The flags can be combined to clear all.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CLEAR_VALUE structure (d3d12.h)

Article02/22/2024

Describes a value used to optimize clear operations for a particular resource.

Syntax

C++

```
typedef struct D3D12_CLEAR_VALUE {
    DXGI_FORMAT Format;
    union {
        FLOAT Color[4];
        D3D12_DEPTH_STENCIL_VALUE DepthStencil;
    };
} D3D12_CLEAR_VALUE;
```

Members

Format

Specifies one member of the [DXGI_FORMAT](#) enum.

The format of the commonly cleared color follows the same validation rules as a view/descriptor creation. In general, the format of the clear color can be any format in the same typeless group that the resource format belongs to.

This *Format* must match the format of the view used during the clear operation. It indicates whether the *Color* or the *DepthStencil* member is valid and how to convert the values for usage with the resource.

Color[4]

Specifies a 4-entry array of float values, determining the RGBA value. The order of RGBA matches the order used with [ClearRenderTargetView](#).

DepthStencil

Specifies one member of [D3D12_DEPTH_STENCIL_VALUE](#). These values match the semantics of *Depth* and *Stencil* in [ClearDepthStencilView](#).

Remarks

This structure is optionally passed into the following methods:

- [ID3D12Device::CreateCommittedResource](#)
- [ID3D12Device::CreatePlacedResource](#)
- [ID3D12Device::CreateReservedResource](#)

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_CLEAR_VALUE](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COLOR_WRITE_ENABLE enumeration (d3d12.h)

Article02/22/2024

Identifies which components of each pixel of a render target are writable during blending.

Syntax

C++

```
typedef enum D3D12_COLOR_WRITE_ENABLE {
    D3D12_COLOR_WRITE_ENABLE_RED = 1,
    D3D12_COLOR_WRITE_ENABLE_GREEN = 2,
    D3D12_COLOR_WRITE_ENABLE_BLUE = 4,
    D3D12_COLOR_WRITE_ENABLE_ALPHA = 8,
    D3D12_COLOR_WRITE_ENABLE_ALL
};
```

Constants

[] Expand table

	<code>D3D12_COLOR_WRITE_ENABLE_RED</code>
Value:	1
	Allow data to be stored in the red component.
	<code>D3D12_COLOR_WRITE_ENABLE_GREEN</code>
Value:	2
	Allow data to be stored in the green component.
	<code>D3D12_COLOR_WRITE_ENABLE_BLUE</code>
Value:	4
	Allow data to be stored in the blue component.
	<code>D3D12_COLOR_WRITE_ENABLE_ALPHA</code>
Value:	8
	Allow data to be stored in the alpha component.
	<code>D3D12_COLOR_WRITE_ENABLE_ALL</code>
	Allow data to be stored in all components.

Remarks

This enum is used by the [D3D12_RENDER_TARGET_BLEND_DESC](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_BLEND_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_LIST_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies flags to be used when creating a command list.

Syntax

C++

```
typedef enum D3D12_COMMAND_LIST_FLAGS {
    D3D12_COMMAND_LIST_FLAG_NONE
};
```

Constants

[+] Expand table

D3D12_COMMAND_LIST_FLAG_NONE

No flags specified.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

[+] Yes

[-] No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_LIST_SUPPORT_FLAGS enumeration (d3d12.h)

Article02/22/2024

Used to determine which kinds of command lists are capable of supporting various operations. For example, whether a command list supports immediate writes.

Syntax

C++

```
typedef enum D3D12_COMMAND_LIST_SUPPORT_FLAGS {
    D3D12_COMMAND_LIST_SUPPORT_FLAG_NONE = 0,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_DIRECT,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_BUNDLE,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_COMPUTE,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_COPY,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_VIDEO_DECODE,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_VIDEO_PROCESS,
    D3D12_COMMAND_LIST_SUPPORT_FLAG_VIDEO_ENCODE
} ;
```

Constants

 Expand table

	<code>D3D12_COMMAND_LIST_SUPPORT_FLAG_NONE</code>
Value:	0
	Specifies that no command list supports the operation in question.
	<code>D3D12_COMMAND_LIST_SUPPORT_FLAG_DIRECT</code>
	Specifies that direct command lists can support the operation in question.
	<code>D3D12_COMMAND_LIST_SUPPORT_FLAG_BUNDLE</code>
	Specifies that command list bundles can support the operation in question.
	<code>D3D12_COMMAND_LIST_SUPPORT_FLAG_COMPUTE</code>
	Specifies that compute command lists can support the operation in question.
	<code>D3D12_COMMAND_LIST_SUPPORT_FLAG_COPY</code>
	Specifies that copy command lists can support the operation in question.

D3D12_COMMAND_LIST_SUPPORT_FLAG_VIDEO_DECODE

Specifies that video-decode command lists can support the operation in question.

D3D12_COMMAND_LIST_SUPPORT_FLAG_VIDEO_PROCESS

Specifies that video-processing command lists can support the operation in question.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[D3D12_COMMAND_LIST_TYPE](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_LIST_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of a command list.

Syntax

C++

```
typedef enum D3D12_COMMAND_LIST_TYPE {
    D3D12_COMMAND_LIST_TYPE_DIRECT = 0,
    D3D12_COMMAND_LIST_TYPE_BUNDLE = 1,
    D3D12_COMMAND_LIST_TYPE_COMPUTE = 2,
    D3D12_COMMAND_LIST_TYPE_COPY = 3,
    D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE = 4,
    D3D12_COMMAND_LIST_TYPE_VIDEO_PROCESS = 5,
    D3D12_COMMAND_LIST_TYPE_VIDEO_ENCODE,
    D3D12_COMMAND_LIST_TYPE_NONE
} ;
```

Constants

[\[\] Expand table](#)

`D3D12_COMMAND_LIST_TYPE_DIRECT`

Value: 0

Specifies a command buffer that the GPU can execute. A direct command list doesn't inherit any GPU state.

`D3D12_COMMAND_LIST_TYPE_BUNDLE`

Value: 1

Specifies a command buffer that can be executed only directly via a direct command list. A bundle command list inherits all GPU state (except for the currently set pipeline state object and primitive topology).

`D3D12_COMMAND_LIST_TYPE_COMPUTE`

Value: 2

Specifies a command buffer for computing.

`D3D12_COMMAND_LIST_TYPE_COPY`

Value: 3

Specifies a command buffer for copying.

D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE

Value: 4

Specifies a command buffer for video decoding.

D3D12_COMMAND_LIST_TYPE_VIDEO_PROCESS

Value: 5

Specifies a command buffer for video processing.

Remarks

This enum is used by the following methods:

- [CreateCommandAllocator](#)
- [CreateCommandQueue](#)
- [CreateCommandList](#)

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_POOL_FLAGS enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_COMMAND_POOL_FLAGS {
    D3D12_COMMAND_POOL_FLAG_NONE
} ;
```

Constants

[] Expand table

D3D12_COMMAND_POOL_FLAG_NONE

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_QUEUE_DESC structure (d3d12.h)

Article02/22/2024

Describes a command queue.

Syntax

C++

```
typedef struct D3D12_COMMAND_QUEUE_DESC {
    D3D12_COMMAND_LIST_TYPE    Type;
    INT                         Priority;
    D3D12_COMMAND_QUEUE_FLAGS Flags;
    UINT                        NodeMask;
} D3D12_COMMAND_QUEUE_DESC;
```

Members

Type

Specifies one member of [D3D12_COMMAND_LIST_TYPE](#).

Priority

The priority for the command queue, as a [D3D12_COMMAND_QUEUE_PRIORITY](#) enumeration constant to select normal or high priority.

Flags

Specifies any flags from the [D3D12_COMMAND_QUEUE_FLAGS](#) enumeration.

NodeMask

For single GPU operation, set this to zero. If there are multiple GPU nodes, set a bit to identify the node (the device's physical adapter) to which the command queue applies. Each bit in the mask corresponds to a single node. Only 1 bit must be set. Refer to [Multi-adapter systems](#).

Remarks

This structure is passed into [CreateCommandQueue](#).

This structure is returned by [ID3D12CommandQueue::GetDesc](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_QUEUE_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies flags to be used when creating a command queue.

Syntax

C++

```
typedef enum D3D12_COMMAND_QUEUE_FLAGS {
    D3D12_COMMAND_QUEUE_FLAG_NONE = 0,
    D3D12_COMMAND_QUEUE_FLAG_DISABLE_GPU_TIMEOUT = 0x1
};
```

Constants

 Expand table

`D3D12_COMMAND_QUEUE_FLAG_NONE`

Value: *0*

Indicates a default command queue.

`D3D12_COMMAND_QUEUE_FLAG_DISABLE_GPU_TIMEOUT`

Value: *0x1*

Indicates that the GPU timeout should be disabled for this command queue.

Remarks

This enum is used by the [D3D12_COMMAND_QUEUE_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_QUEUE_PRIORITY enumeration (d3d12.h)

Article02/22/2024

Defines priority levels for a command queue.

Syntax

C++

```
typedef enum D3D12_COMMAND_QUEUE_PRIORITY {
    D3D12_COMMAND_QUEUE_PRIORITY_NORMAL = 0,
    D3D12_COMMAND_QUEUE_PRIORITY_HIGH = 100,
    D3D12_COMMAND_QUEUE_PRIORITY_GLOBAL_REALTIME = 10000
} ;
```

Constants

 Expand table

D3D12_COMMAND_QUEUE_PRIORITY_NORMAL

Value: 0

Normal priority.

D3D12_COMMAND_QUEUE_PRIORITY_HIGH

Value: 100

High priority.

D3D12_COMMAND_QUEUE_PRIORITY_GLOBAL_REALTIME

Value: 10000

Global realtime priority.

Remarks

This enumeration is used by the **Priority** member of the [D3D12_COMMAND_QUEUE_DESC](#) structure.

An application must be sufficiently privileged in order to create a command queue that has global realtime priority. If the application is not sufficiently privileged or if neither

the adapter or driver can provide the necessary preemption, then requests to create a global realtime priority queue fail; such a failure could be due to a lack of hardware support or due to conflicts with other command queue parameters. Requests to create a global realtime command queue won't silently downgrade the priority when it can't be supported; the request succeeds or fails as-is to indicate to the application whether or not the command queue is guaranteed to execute before any other queue.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_RECORDER_FLAGS enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_COMMAND_RECORDER_FLAGS {
    D3D12_COMMAND_RECORDER_FLAG_NONE
} ;
```

Constants

[] Expand table

D3D12_COMMAND_RECORDER_FLAG_NONE

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMMAND_SIGNATURE_DESC structure (d3d12.h)

Article02/22/2024

Describes the arguments (parameters) of a command signature.

Syntax

C++

```
typedef struct D3D12_COMMAND_SIGNATURE_DESC {
    UINT             ByteStride;
    UINT             NumArgumentDescs;
    const D3D12_INDIRECT_ARGUMENT_DESC *pArgumentDescs;
    UINT             NodeMask;
} D3D12_COMMAND_SIGNATURE_DESC;
```

Members

`ByteStride`

Specifies the size of each command in the drawing buffer, in bytes.

`NumArgumentDescs`

Specifies the number of arguments in the command signature.

`pArgumentDescs`

An array of `D3D12_INDIRECT_ARGUMENT_DESC` structures, containing details of the arguments, including whether the argument is a vertex buffer, constant, constant buffer view, shader resource view, or unordered access view.

`NodeMask`

For single GPU operation, set this to zero. If there are multiple GPU nodes, set bits to identify the nodes (the device's physical adapters) for which the command signature is to apply. Each bit in the mask corresponds to a single node. Refer to [Multi-adapter systems](#).

Remarks

Use this structure by [CreateCommandSignature](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_COMPARISON_FUNC enumeration (d3d12.h)

Article 02/14/2023

Specifies comparison options.

Syntax

C++

```
typedef enum D3D12_COMPARISON_FUNC {
    D3D12_COMPARISON_FUNC_NONE,
    D3D12_COMPARISON_FUNC_NEVER = 1,
    D3D12_COMPARISON_FUNC_LESS = 2,
    D3D12_COMPARISON_FUNC_EQUAL = 3,
    D3D12_COMPARISON_FUNC_LESS_EQUAL = 4,
    D3D12_COMPARISON_FUNC_GREATER = 5,
    D3D12_COMPARISON_FUNC_NOT_EQUAL = 6,
    D3D12_COMPARISON_FUNC_GREATER_EQUAL = 7,
    D3D12_COMPARISON_FUNC_ALWAYS = 8
} ;
```

Constants

[] Expand table

<code>D3D12_COMPARISON_FUNC_NEVER</code>
--

Value: 1

Never pass the comparison.

<code>D3D12_COMPARISON_FUNC_LESS</code>

Value: 2

If the source data is less than the destination data, the comparison passes.

<code>D3D12_COMPARISON_FUNC_EQUAL</code>
--

Value: 3

If the source data is equal to the destination data, the comparison passes.

<code>D3D12_COMPARISON_FUNC_LESS_EQUAL</code>

Value: 4

If the source data is less than or equal to the destination data, the comparison passes.

`D3D12_COMPARISON_FUNC_GREATER`

Value: 5

If the source data is greater than the destination data, the comparison passes.

`D3D12_COMPARISON_FUNC_NOT_EQUAL`

Value: 6

If the source data is not equal to the destination data, the comparison passes.

`D3D12_COMPARISON_FUNC_GREATER_EQUAL`

Value: 7

If the source data is greater than or equal to the destination data, the comparison passes.

`D3D12_COMPARISON_FUNC_ALWAYS`

Value: 8

Always pass the comparison.

Remarks

A comparison option determines how the runtime compares source (new) data against destination (existing) data before storing the new data. The comparison option is declared in a description before an object is created. The API allows you to set a comparison option for

- a depth-stencil buffer ([D3D12_DEPTH_STENCIL_DESC](#))
- depth-stencil operations ([D3D12_DEPTH_STENCILOP_DESC](#))
- sampler state ([D3D12_SAMPLER_DESC](#))

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_COMPUTE_PIPELINE_STATE_DESC structure (d3d12.h)

Article02/22/2024

Describes a compute pipeline state object.

Syntax

C++

```
typedef struct D3D12_COMPUTE_PIPELINE_STATE_DESC {
    ID3D12RootSignature          *pRootSignature;
    D3D12_SHADER_BYTECODE        CS;
    UINT                          NodeMask;
    D3D12_CACHED_PIPELINE_STATE CachedPSO;
    D3D12_PIPELINE_STATE_FLAGS   Flags;
} D3D12_COMPUTE_PIPELINE_STATE_DESC;
```

Members

pRootSignature

A pointer to the [ID3D12RootSignature](#) object.

CS

A [D3D12_SHADER_BYTECODE](#) structure that describes the compute shader.

NodeMask

For single GPU operation, set this to zero. If there are multiple GPU nodes, set bits to identify the nodes (the device's physical adapters) for which the compute pipeline state is to apply. Each bit in the mask corresponds to a single node. Refer to [Multi-adapter systems](#).

CachedPSO

A cached pipeline state object, as a [D3D12_CACHED_PIPELINE_STATE](#) structure. pCachedBlob and CachedBlobSizeInBytes may be set to NULL and 0 respectively.

Flags

A [D3D12_PIPELINE_STATE_FLAGS](#) enumeration constant such as for "tool debug".

Remarks

This structure is used by [CreateComputePipelineState](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CONSERVATIVE_RASTERIZATION_MODE enumeration (d3d12.h)

Article 02/22/2024

Identifies whether conservative rasterization is on or off.

Syntax

C++

```
typedef enum D3D12_CONSERVATIVE_RASTERIZATION_MODE {
    D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF = 0,
    D3D12_CONSERVATIVE_RASTERIZATION_MODE_ON = 1
};
```

Constants

[+] Expand table

<code>D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF</code>
--

Value: 0

Conservative rasterization is off.

<code>D3D12_CONSERVATIVE_RASTERIZATION_MODE_ON</code>

Value: 1

Conservative rasterization is on.

Remarks

This enum is used by the [D3D12_RASTERIZER_DESC](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Conservative Rasterization](#)

[Core Enumerations](#)

[D3D12_CONSERVATIVE_RASTERIZATION_TIER](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CONSERVATIVE_RASTERIZATION_TIER enumeration (d3d12.h)

Article 02/22/2024

Identifies the tier level of conservative rasterization.

Syntax

C++

```
typedef enum D3D12_CONSERVATIVE_RASTERIZATION_TIER {
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_NOT_SUPPORTED = 0,
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_1 = 1,
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_2 = 2,
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_3 = 3
};
```

Constants

[] [Expand table](#)

<code>D3D12_CONSERVATIVE_RASTERIZATION_TIER_NOT_SUPPORTED</code>
--

Value: 0

Conservative rasterization is not supported.

<code>D3D12_CONSERVATIVE_RASTERIZATION_TIER_1</code>
--

Value: 1

Tier 1 enforces a maximum 1/2 pixel uncertainty region and does not support post-snap degenerates. This is good for tiled rendering, a texture atlas, light map generation and sub-pixel shadow maps.

<code>D3D12_CONSERVATIVE_RASTERIZATION_TIER_2</code>
--

Value: 2

Tier 2 reduces the maximum uncertainty region to 1/256 and requires post-snap degenerates not be culled. This tier is helpful for CPU-based algorithm acceleration (such as voxelization).

<code>D3D12_CONSERVATIVE_RASTERIZATION_TIER_3</code>
--

Value: 3

Tier 3 maintains a maximum 1/256 uncertainty region and adds support for inner input coverage. Inner input coverage adds the new value `SV_InnerCoverage` to High Level Shading Language (HLSL). This is a 32-bit scalar integer that can be specified on input to a pixel shader, and

represents the underestimated conservative rasterization information (that is, whether a pixel is guaranteed-to-be-fully covered). This tier is helpful for occlusion culling.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Conservative Rasterization](#)

[Core Enumerations](#)

[D3D12_CONSERVATIVE_RASTERIZATION_MODE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CONSTANT_BUFFER_VIEW_DESC structure (d3d12.h)

Article 02/22/2024

Describes a constant buffer to view.

Syntax

C++

```
typedef struct D3D12_CONSTANT_BUFFER_VIEW_DESC {
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation;
    UINT                     SizeInBytes;
} D3D12_CONSTANT_BUFFER_VIEW_DESC;
```

Members

`BufferLocation`

The `D3D12_GPU_VIRTUAL_ADDRESS` of the constant buffer.
`D3D12_GPU_VIRTUAL_ADDRESS` is a `typedef'd` alias of `UINT64`.

`SizeInBytes`

The size in bytes of the constant buffer.

Remarks

This structure is used by [CreateConstantBufferView](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CPU_DESCRIPTOR_HANDLE structure (d3d12.h)

Article 02/22/2024

Describes a CPU descriptor handle.

Syntax

C++

```
typedef struct D3D12_CPU_DESCRIPTOR_HANDLE {
    SIZE_T ptr;
} D3D12_CPU_DESCRIPTOR_HANDLE;
```

Members

ptr

The address of the descriptor.

Remarks

This structure is returned by the following methods:

- [ID3D12DescriptorHeap::GetCPUDescriptorHandleForHeapStart](#)

This structure is passed into the following methods:

- [ID3D12Device::CopyDescriptors](#)
- [ID3D12Device::CopyDescriptorsSimple](#)
- [ID3D12Device::CreateConstantBufferView](#)
- [ID3D12Device::CreateShaderResourceView](#)
- [ID3D12Device::CreateUnorderedAccessView](#)
- [ID3D12Device::CreateRenderTargetView](#)
- [ID3D12Device::CreateDepthStencilView](#)
- [ID3D12Device::CreateSampler](#)
- [ID3D12GraphicsCommandList::ClearDepthStencilView](#)
- [ID3D12GraphicsCommandList::ClearRenderTargetView](#)
- [ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint](#)

- ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat
- ID3D12GraphicsCommandList::OMSetRenderTargets

To get the handle increment size use [ID3D12Device.GetDescriptorHandleIncrementSize](#)

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_CPU_DESCRIPTOR_HANDLE](#) [ID3D12Device.GetDescriptorHandleIncrementSize](#)

[Core Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CPU_PAGE_PROPERTY enumeration (d3d12.h)

Article02/22/2024

Specifies the CPU-page properties for the heap.

Syntax

C++

```
typedef enum D3D12_CPU_PAGE_PROPERTY {
    D3D12_CPU_PAGE_PROPERTY_UNKNOWN = 0,
    D3D12_CPU_PAGE_PROPERTY_NOT_AVAILABLE = 1,
    D3D12_CPU_PAGE_PROPERTY_WRITE_COMBINE = 2,
    D3D12_CPU_PAGE_PROPERTY_WRITE_BACK = 3
};
```

Constants

[] Expand table

`D3D12_CPU_PAGE_PROPERTY_UNKNOWN`

Value: 0

The CPU-page property is unknown.

`D3D12_CPU_PAGE_PROPERTY_NOT_AVAILABLE`

Value: 1

The CPU cannot access the heap, therefore no page properties are available.

`D3D12_CPU_PAGE_PROPERTY_WRITE_COMBINE`

Value: 2

The CPU-page property is write-combined.

`D3D12_CPU_PAGE_PROPERTY_WRITE_BACK`

Value: 3

The CPU-page property is write-back.

Remarks

This enum is used by the [D3D12_HEAP_PROPERTIES](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_CROSS_NODE_SHARING_TIER enumeration (d3d12.h)

Article02/22/2024

Specifies the level of sharing across nodes of an adapter, such as Tier 1 Emulated, Tier 1, or Tier 2.

Syntax

C++

```
typedef enum D3D12_CROSS_NODE_SHARING_TIER {
    D3D12_CROSS_NODE_SHARING_TIER_NOT_SUPPORTED = 0,
    D3D12_CROSS_NODE_SHARING_TIER_1_EMULATED = 1,
    D3D12_CROSS_NODE_SHARING_TIER_1 = 2,
    D3D12_CROSS_NODE_SHARING_TIER_2 = 3,
    D3D12_CROSS_NODE_SHARING_TIER_3 = 4
} ;
```

Constants

[+] Expand table

D3D12_CROSS_NODE_SHARING_TIER_NOT_SUPPORTED

Value: 0

If an adapter has only 1 node, then cross-node sharing doesn't apply, so the **CrossNodeSharingTier** member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure is set to D3D12_CROSS_NODE_SHARING_NOT_SUPPORTED.

D3D12_CROSS_NODE_SHARING_TIER_1_EMULATED

Value: 1

Tier 1 Emulated. Devices that set the **CrossNodeSharingTier** member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure to D3D12_CROSS_NODE_SHARING_TIER_1_EMULATED have Tier 1 support. However, drivers stage these copy operations through a driver-internal system memory allocation. This will cause these copy operations to consume time on the destination GPU as well as the source.

D3D12_CROSS_NODE_SHARING_TIER_1

Value: 2

Tier 1. Devices that set the **CrossNodeSharingTier** member of the

[D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure to D3D12_CROSS_NODE_SHARING_TIER_1 only support the following cross-node copy operations:

- [ID3D12CommandList::CopyBufferRegion](#)
- [ID3D12CommandList::CopyTextureRegion](#)
- [ID3D12CommandList::CopyResource](#)

Additionally, the cross-node resource must be the destination of the copy operation.

`D3D12_CROSS_NODE_SHARING_TIER_2`

Value: 3

Tier 2. Devices that set the `CrossNodeSharingTier` member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure to D3D12_CROSS_NODE_SHARING_TIER_2 support all operations across nodes, except for the following:

- Render target views.
- Depth stencil views.
- UAV atomic operations. Similar to CPU/GPU interop, shaders may perform UAV atomic operations; however, no atomicity across adapters is guaranteed.

Applications can retrieve the node where a resource/heap exists from the [D3D12_HEAP_DESC](#) structure. These values are retrievable for opened resources. The runtime performs the appropriate re-mapping in case the 2 devices are using different UMD-specified node re-mappings.

`D3D12_CROSS_NODE_SHARING_TIER_3`

Value: 4

Indicates support for [D3D12_HEAP_FLAG_ALLOW_SHADER_ATOMICS](#) on heaps that are visible to multiple nodes.

Remarks

This enum is used by the `CrossNodeSharingTier` member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_CULL_MODE enumeration (d3d12.h)

Article02/22/2024

Specifies triangles facing a particular direction are not drawn.

Syntax

C++

```
typedef enum D3D12_CULL_MODE {
    D3D12_CULL_MODE_NONE = 1,
    D3D12_CULL_MODE_FRONT = 2,
    D3D12_CULL_MODE_BACK = 3
};
```

Constants

[+] Expand table

`D3D12_CULL_MODE_NONE`

Value: 1

Always draw all triangles.

`D3D12_CULL_MODE_FRONT`

Value: 2

Do not draw triangles that are front-facing.

`D3D12_CULL_MODE_BACK`

Value: 3

Do not draw triangles that are back-facing.

Remarks

Cull mode is specified in a [D3D12_RASTERIZER_DESC](#) structure.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RASTERIZER_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_FILTER_REDUCTION macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_FILTER_REDUCTION(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_IS_ANISOTROPIC_FILTER macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_IS_ANISOTROPIC_FILTER(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_IS_COMPARISON_FILTER macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_IS_COMPARISON_FILTER(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_MAG_FILTER macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_MAG_FILTER(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_MIN_FILTER macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_MIN_FILTER(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_MIP_FILTER macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_MIP_FILTER(  
    D3D12Filter  
);
```

Parameters

D3D12Filter

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DECODE_SHADER_4_COMPONENT_MAPPING macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_DECODE_SHADER_4_COMPONENT_MAPPING(
    ComponentToExtract,
    Mapping
);
```

Parameters

ComponentToExtract

Mapping

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_STENCIL_DESC structure (d3d12.h)

Article 07/27/2022

Describes depth-stencil state.

Syntax

C++

```
typedef struct D3D12_DEPTH_STENCIL_DESC {
    BOOL DepthEnable;
    D3D12_DEPTH_WRITE_MASK DepthWriteMask;
    D3D12_COMPARISON_FUNC DepthFunc;
    BOOL StencilEnable;
    UINT8 StencilReadMask;
    UINT8 StencilWriteMask;
    D3D12_DEPTH_STENCILOP_DESC FrontFace;
    D3D12_DEPTH_STENCILOP_DESC BackFace;
} D3D12_DEPTH_STENCIL_DESC;
```

Members

DepthEnable

Specifies whether to enable depth testing. Set this member to **TRUE** to enable depth testing.

DepthWriteMask

A [D3D12_DEPTH_WRITE_MASK](#)-typed value that identifies a portion of the depth-stencil buffer that can be modified by depth data.

DepthFunc

A [D3D12_COMPARISON_FUNC](#)-typed value that identifies a function that compares depth data against existing depth data.

StencilEnable

Specifies whether to enable stencil testing. Set this member to **TRUE** to enable stencil testing.

StencilReadMask

Identify a portion of the depth-stencil buffer for reading stencil data.

StencilWriteMask

Identify a portion of the depth-stencil buffer for writing stencil data.

FrontFace

A [D3D12_DEPTH_STENCILOP_DESC](#) structure that describes how to use the results of the depth test and the stencil test for pixels whose surface normal is facing towards the camera.

BackFace

A [D3D12_DEPTH_STENCILOP_DESC](#) structure that describes how to use the results of the depth test and the stencil test for pixels whose surface normal is facing away from the camera.

Remarks

A [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) object contains a depth-stencil-state structure that controls how depth-stencil testing is performed by the output-merger stage.

This table shows the default values of depth-stencil states.

[+] Expand table

State	Default Value
DepthEnable	TRUE
DepthWriteMask	D3D12_DEPTH_WRITE_MASK_ALL
DepthFunc	D3D12_COMPARISON_FUNC_LESS
StencilEnable	FALSE
StencilReadMask	D3D12_DEFAULT_STENCIL_READ_MASK
StencilWriteMask	D3D12_DEFAULT_STENCIL_WRITE_MASK
FrontFace.StencilFailOp and	D3D12_STENCIL_OP_KEEP

BackFace.StencilFailOp	
FrontFace.StencilDepthFailOp and	D3D12_STENCIL_OP_KEEP
BackFace.StencilDepthFailOp	
FrontFace.StencilPassOp and	D3D12_STENCIL_OP_KEEP
BackFace.StencilPassOp	
FrontFace.StencilFunc and	D3D12_COMPARISON_FUNC_ALWAYS
BackFace.StencilFunc	

The formats that support stenciling are DXGI_FORMAT_D24_UNORM_S8_UINT and DXGI_FORMAT_D32_FLOAT_S8X24_UINT.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_STENCIL_DESC1 structure (d3d12.h)

Article 02/22/2024

Describes depth-stencil state.

Syntax

C++

```
typedef struct D3D12_DEPTH_STENCIL_DESC1 {
    BOOL DepthEnable;
    D3D12_DEPTH_WRITE_MASK DepthWriteMask;
    D3D12_COMPARISON_FUNC DepthFunc;
    BOOL StencilEnable;
    UINT8 StencilReadMask;
    UINT8 StencilWriteMask;
    D3D12_DEPTH_STENCILOP_DESC FrontFace;
    D3D12_DEPTH_STENCILOP_DESC BackFace;
    BOOL DepthBoundsTestEnable;
} D3D12_DEPTH_STENCIL_DESC1;
```

Members

DepthEnable

Specifies whether to enable depth testing. Set this member to **TRUE** to enable depth testing.

DepthWriteMask

A [D3D12_DEPTH_WRITE_MASK](#)-typed value that identifies a portion of the depth-stencil buffer that can be modified by depth data.

DepthFunc

A [D3D12_COMPARISON_FUNC](#)-typed value that identifies a function that compares depth data against existing depth data.

StencilEnable

Specifies whether to enable stencil testing. Set this member to **TRUE** to enable stencil testing.

StencilReadMask

Identify a portion of the depth-stencil buffer for reading stencil data.

StencilWriteMask

Identify a portion of the depth-stencil buffer for writing stencil data.

FrontFace

A [D3D12_DEPTH_STENCILOP_DESC](#) structure that describes how to use the results of the depth test and the stencil test for pixels whose surface normal is facing towards the camera.

BackFace

A [D3D12_DEPTH_STENCILOP_DESC](#) structure that describes how to use the results of the depth test and the stencil test for pixels whose surface normal is facing away from the camera.

DepthBoundsTestEnable

TRUE to enable depth-bounds testing; otherwise, FALSE. The default value is FALSE.

Remarks

A [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) object contains a depth-stencil-state structure that controls how depth-stencil testing is performed by the output-merger stage.

This table shows the default values of depth-stencil states.

[] [Expand table](#)

State	Default Value
DepthEnable	TRUE
DepthWriteMask	D3D12_DEPTH_WRITE_MASK_ALL
DepthFunc	D3D12_COMPARISON_LESS
StencilEnable	FALSE
StencilReadMask	D3D12_DEFAULT_STENCIL_READ_MASK
StencilWriteMask	D3D12_DEFAULT_STENCIL_WRITE_MASK

FrontFace.StencilFunc and	D3D12_COMPARISON_ALWAYS
BackFace.StencilFunc	
FrontFace.StencilDepthFailOp and	D3D12_STENCIL_OP_KEEP
BackFace.StencilDepthFailOp	
FrontFace.StencilPassOp and	D3D12_STENCIL_OP_KEEP
BackFace.StencilPassOp	
FrontFace.StencilFailOp and	D3D12_STENCIL_OP_KEEP
BackFace.StencilFailOp	
DepthBoundsTestEnable	FALSE

The formats that support stenciling are DXGI_FORMAT_D24_UNORM_S8_UINT and DXGI_FORMAT_D32_FLOAT_S8X24_UINT.

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_STENCIL_VALUE structure (d3d12.h)

Article02/22/2024

Specifies a depth and stencil value.

Syntax

C++

```
typedef struct D3D12_DEPTH_STENCIL_VALUE {
    FLOAT Depth;
    UINT8 Stencil;
} D3D12_DEPTH_STENCIL_VALUE;
```

Members

Depth

Specifies the depth value.

Stencil

Specifies the stencil value.

Remarks

This structure is used in the [D3D12_CLEAR_VALUE](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_STENCIL_VIEW_DESC structure (d3d12.h)

Article04/02/2021

Describes the subresources of a texture that are accessible from a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_DEPTH_STENCIL_VIEW_DESC {
    DXGI_FORMAT          Format;
    D3D12_DSV_DIMENSION ViewDimension;
    D3D12_DSV_FLAGS      Flags;
    union {
        D3D12_TEX1D_DSV      Texture1D;
        D3D12_TEX1D_ARRAY_DSV Texture1DArray;
        D3D12_TEX2D_DSV      Texture2D;
        D3D12_TEX2D_ARRAY_DSV Texture2DArray;
        D3D12_TEX2DMS_DSV    Texture2DMS;
        D3D12_TEX2DMS_ARRAY_DSV Texture2DMSArray;
    };
} D3D12_DEPTH_STENCIL_VIEW_DESC;
```

Members

Format

A [DXGI_FORMAT](#)-typed value that specifies the viewing format. For allowable formats, see Remarks.

ViewDimension

A [D3D12_DSV_DIMENSION](#)-typed value that specifies how the depth-stencil resource will be accessed. This member also determines which _DSV to use in the following union.

Flags

A combination of [D3D12_DSV_FLAGS](#) enumeration constants that are combined by using a bitwise OR operation. The resulting value specifies whether the texture is read only.

Pass 0 to specify that it isn't read only; otherwise, pass one or more of the members of the **D3D12_DSV_FLAGS** enumerated type.

Texture1D

A [D3D12_TEX1D_DSV](#) structure that specifies a 1D texture subresource.

Texture1DArray

A [D3D12_TEX1D_ARRAY_DSV](#) structure that specifies an array of 1D texture subresources.

Texture2D

A [D3D12_TEX2D_DSV](#) structure that specifies a 2D texture subresource.

Texture2DArray

A [D3D12_TEX2D_ARRAY_DSV](#) structure that specifies an array of 2D texture subresources.

Texture2DMS

A [D3D12_TEX2DMS_DSV](#) structure that specifies a multisampled 2D texture.

Texture2DMSArray

A [D3D12_TEX2DMS_ARRAY_DSV](#) structure that specifies an array of multisampled 2D textures.

Remarks

These are valid formats for a depth-stencil view:

- DXGI_FORMAT_D16_UNORM
- DXGI_FORMAT_D24_UNORM_S8_UINT
- DXGI_FORMAT_D32_FLOAT
- DXGI_FORMAT_D32_FLOAT_S8X24_UINT
- DXGI_FORMAT_UNKNOWN

A depth-stencil view can't use a typeless format. If the format chosen is DXGI_FORMAT_UNKNOWN, the format of the parent resource is used.

Pass a depth-stencil-view description into [ID3D12Device::CreateDepthStencilView](#) to create a depth-stencil view.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_STENCILOP_DESC structure (d3d12.h)

Article02/22/2024

Describes stencil operations that can be performed based on the results of stencil test.

Syntax

C++

```
typedef struct D3D12_DEPTH_STENCILOP_DESC {
    D3D12_STENCIL_OP      StencilFailOp;
    D3D12_STENCIL_OP      StencilDepthFailOp;
    D3D12_STENCIL_OP      StencilPassOp;
    D3D12_COMPARISON_FUNC StencilFunc;
} D3D12_DEPTH_STENCILOP_DESC;
```

Members

`StencilFailOp`

A [D3D12_STENCIL_OP](#)-typed value that identifies the stencil operation to perform when stencil testing fails.

`StencilDepthFailOp`

A [D3D12_STENCIL_OP](#)-typed value that identifies the stencil operation to perform when stencil testing passes and depth testing fails.

`StencilPassOp`

A [D3D12_STENCIL_OP](#)-typed value that identifies the stencil operation to perform when stencil testing and depth testing both pass.

`StencilFunc`

A [D3D12_COMPARISON_FUNC](#)-typed value that identifies the function that compares stencil data against existing stencil data.

Remarks

All stencil operations are specified as a [D3D12_STENCIL_OP](#)-typed value. Each stencil operation can be set differently based on the outcome of the stencil test, which is referred to as **StencilFunc**, in the stencil test portion of depth-stencil testing.

Members of [D3D12_DEPTH_STENCIL_DESC](#) have this structure for their data type.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEPTH_WRITE_MASK enumeration (d3d12.h)

Article02/22/2024

Identifies the portion of a depth-stencil buffer for writing depth data.

Syntax

C++

```
typedef enum D3D12_DEPTH_WRITE_MASK {
    D3D12_DEPTH_WRITE_MASK_ZERO = 0,
    D3D12_DEPTH_WRITE_MASK_ALL = 1
};
```

Constants

[+] Expand table

D3D12_DEPTH_WRITE_MASK_ZERO

Value: 0

Turn off writes to the depth-stencil buffer.

D3D12_DEPTH_WRITE_MASK_ALL

Value: 1

Turn on writes to the depth-stencil buffer.

Remarks

This enum is used by the [D3D12_DEPTH_STENCIL_DESC](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_DESCRIPTOR_HEAP_DESC structure (d3d12.h)

Article02/22/2024

Describes the descriptor heap.

Syntax

C++

```
typedef struct D3D12_DESCRIPTOR_HEAP_DESC {
    D3D12_DESCRIPTOR_HEAP_TYPE Type;
    UINT                      NumDescriptors;
    D3D12_DESCRIPTOR_HEAP_FLAGS Flags;
    UINT                      NodeMask;
} D3D12_DESCRIPTOR_HEAP_DESC;
```

Members

Type

A [D3D12_DESCRIPTOR_HEAP_TYPE](#)-typed value that specifies the types of descriptors in the heap.

NumDescriptors

The number of descriptors in the heap.

Flags

A combination of [D3D12_DESCRIPTOR_HEAP_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for the heap.

NodeMask

For single-adapter operation, set this to zero. If there are multiple adapter nodes, set a bit to identify the node (one of the device's physical adapters) to which the descriptor heap applies. Each bit in the mask corresponds to a single node. Only one bit must be set. See [Multi-adapter systems](#).

Remarks

This structure is used by the following:

- [ID3D12DescriptorHeap::GetDesc](#)
- [ID3D12Device::CreateDescriptorHeap](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_HEAP_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies options for a heap.

Syntax

C++

```
typedef enum D3D12_DESCRIPTOR_HEAP_FLAGS {
    D3D12_DESCRIPTOR_HEAP_FLAG_NONE = 0,
    D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE = 0x1
};
```

Constants

[] Expand table

<code>D3D12_DESCRIPTOR_HEAP_FLAG_NONE</code>
--

Value: 0

Indicates default usage of a heap.

<code>D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE</code>
--

Value: 0x1

The flag `D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE` can optionally be set on a descriptor heap to indicate it is bound on a command list for reference by shaders. Descriptor heaps created *without* this flag allow applications the option to stage descriptors in CPU memory before copying them to a shader visible descriptor heap, as a convenience. But it is also fine for applications to directly create descriptors into shader visible descriptor heaps with no requirement to stage anything on the CPU.

Descriptor heaps bound via `ID3D12GraphicsCommandList::SetDescriptorHeaps` must have the `D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE` flag set, else the debug layer will produce an error.

Descriptor heaps with the `D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE` flag can't be used as the source heaps in calls to `ID3D12Device::CopyDescriptors` or `ID3D12Device::CopyDescriptorsSimple`, because they could be resident in `WRITE_COMBINE` memory or GPU-local memory, which is very inefficient to read from.

This flag only applies to CBV/SRV/UAV descriptor heaps, and sampler descriptor heaps. It does not apply to other descriptor heap types since shaders do not directly reference the other types. Attempting to create an RTV/DSV heap with `D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE` results in a debug layer error.

Remarks

This enum is used by the [D3D12_DESCRIPTOR_HEAP_DESC](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_HEAP_TYPE enumeration (d3d12.h)

Article01/31/2022

Specifies a type of descriptor heap.

Syntax

C++

```
typedef enum D3D12_DESCRIPTOR_HEAP_TYPE {
    D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV = 0,
    D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER,
    D3D12_DESCRIPTOR_HEAP_TYPE_RTV,
    D3D12_DESCRIPTOR_HEAP_TYPE_DSV,
    D3D12_DESCRIPTOR_HEAP_TYPE_NUM_TYPES
};
```

Constants

[+] Expand table

D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV

Value: 0

The descriptor heap for the combination of constant-buffer, shader-resource, and unordered-access views.

D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER

The descriptor heap for the sampler.

D3D12_DESCRIPTOR_HEAP_TYPE_RTV

The descriptor heap for the render-target view.

D3D12_DESCRIPTOR_HEAP_TYPE_DSV

The descriptor heap for the depth-stencil view.

D3D12_DESCRIPTOR_HEAP_TYPE_NUM_TYPES

The number of types of descriptor heaps.

Remarks

This enum is used by the [D3D12_DESCRIPTOR_HEAP_DESC](#) structure, and the following methods:

- [CopyDescriptors](#)
- [CopyDescriptorsSimple](#)
- [GetDescriptorHandleIncrementSize](#)

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_RANGE structure (d3d12.h)

Article 02/22/2024

Describes a descriptor range.

Syntax

C++

```
typedef struct D3D12_DESCRIPTOR_RANGE {
    D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
    UINT                      NumDescriptors;
    UINT                      BaseShaderRegister;
    UINT                      RegisterSpace;
    UINT                      OffsetInDescriptorsFromTableStart;
} D3D12_DESCRIPTOR_RANGE;
```

Members

RangeType

A [D3D12_DESCRIPTOR_RANGE_TYPE](#)-typed value that specifies the type of descriptor range.

NumDescriptors

The number of descriptors in the range. Use -1 or `UINT_MAX` to specify an unbounded size. If a given descriptor range is unbounded, then it must either be the last range in the table definition, or else the following range in the table definition must have a value for `OffsetInDescriptorsFromTableStart` that is not [D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND](#).

BaseShaderRegister

The base shader register in the range. For example, for shader-resource views (SRVs), 3 maps to ": register(t3); " in HLSL.

RegisterSpace

The register space. Can typically be 0, but allows multiple descriptor arrays of unknown size to not appear to overlap. For example, for SRVs, by extending the example in the

BaseShaderRegister member description, 5 maps to ": register(t3,space5);" in HLSL.

OffsetInDescriptorsFromTableStart

The offset in descriptors, from the start of the descriptor table which was set as the root argument value for this parameter slot. This value can be **D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND**, which indicates this range should immediately follow the preceding range.

Remarks

This structure is a member of the [D3D12_ROOT_DESCRIPTOR_TABLE](#) structure.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_DESCRIPTOR_RANGE](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_RANGE_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies the volatility of both descriptors and the data they reference in a Root Signature 1.1 description, which can enable some driver optimizations.

Syntax

C++

```
typedef enum D3D12_DESCRIPTOR_RANGE_FLAGS {
    D3D12_DESCRIPTOR_RANGE_FLAG_NONE = 0,
    D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_VOLATILE = 0x1,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_VOLATILE = 0x2,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE = 0x4,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC = 0x8,

    D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_STATIC_KEEPING_BUFFER_BOUNDS_CHECKS
    = 0x10000
} ;
```

Constants

[Expand table](#)

D3D12_DESCRIPTOR_RANGE_FLAG_NONE

Value: 0

Default behavior. Descriptors are static, and default assumptions are made for data (for SRV/CBV: DATA_STATIC_WHILE_SET_AT_EXECUTE, and for UAV: DATA_VOLATILE).

D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_VOLATILE

Value: 0x1

If this is the only flag set, then descriptors are volatile and default assumptions are made about data (for SRV/CBV: DATA_STATIC_WHILE_SET_AT_EXECUTE, and for UAV: DATA_VOLATILE).

If this flag is combined with DATA_VOLATILE, then both descriptors and data are volatile, which is equivalent to Root Signature Version 1.0.

If this flag is combined with DATA_STATIC_WHILE_SET_AT_EXECUTE, then descriptors are volatile. This still doesn't allow them to change during command list execution so it is valid to combine the

additional declaration that data is static while set via root descriptor table during execution – the underlying descriptors are effectively static for longer than the data is being promised to be static.

`D3D12_DESCRIPTOR_RANGE_FLAG_DATA_VOLATILE`

Value: *0x2*

Descriptors are static and the data is volatile.

`D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE`

Value: *0x4*

Descriptors are static and data is static while set at execute.

`D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC`

Value: *0x8*

Both descriptors and data are static. This maximizes the potential for driver optimization.

`D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_STATIC_KEEPING_BUFFER_BOUNDS_CHECKS`

Value: *0x10000*

Provides the same benefits as static descriptors (see `D3D12_DESCRIPTOR_RANGE_FLAG_NONE`), except that the driver is not allowed to promote buffers to root descriptors as an optimization, because they must maintain bounds checks and root descriptors do not have those.

Remarks

This enum is used by the [D3D12_DESCRIPTOR_RANGE1](#) structure.

To specify the volatility of just the data referenced by descriptors, refer to [D3D12_ROOT_DESCRIPTOR_FLAGS](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_RANGE_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies a range so that, for example, if part of a descriptor table has 100 shader-resource views (SRVs) that range can be declared in one entry rather than 100.

Syntax

C++

```
typedef enum D3D12_DESCRIPTOR_RANGE_TYPE {
    D3D12_DESCRIPTOR_RANGE_TYPE_SRV = 0,
    D3D12_DESCRIPTOR_RANGE_TYPE_UAV,
    D3D12_DESCRIPTOR_RANGE_TYPE_CBV,
    D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER
};
```

Constants

[] Expand table

	D3D12_DESCRIPTOR_RANGE_TYPE_SRV
Value:	0
	Specifies a range of SRVs.
	D3D12_DESCRIPTOR_RANGE_TYPE_UAV
	Specifies a range of unordered-access views (UAVs).
	D3D12_DESCRIPTOR_RANGE_TYPE_CBV
	Specifies a range of constant-buffer views (CBVs).
	D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER
	Specifies a range of samplers.

Remarks

This enum is used by the [D3D12_DESCRIPTOR_RANGE](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_DESCRIPTOR_RANGE1 structure (d3d12.h)

Article 02/22/2024

Describes a descriptor range, with flags to determine their volatility.

Syntax

C++

```
typedef struct D3D12_DESCRIPTOR_RANGE1 {
    D3D12_DESCRIPTOR_RANGE_TYPE RangeType;
    UINT NumDescriptors;
    UINT BaseShaderRegister;
    UINT RegisterSpace;
    D3D12_DESCRIPTOR_RANGE_FLAGS Flags;
    UINT OffsetInDescriptorsFromTableStart;
} D3D12_DESCRIPTOR_RANGE1;
```

Members

RangeType

A [D3D12_DESCRIPTOR_RANGE_TYPE](#)-typed value that specifies the type of descriptor range.

NumDescriptors

The number of descriptors in the range. Use -1 or `UINT_MAX` to specify unbounded size. Only the last entry in a table can have unbounded size.

BaseShaderRegister

The base shader register in the range. For example, for shader-resource views (SRVs), 3 maps to ": register(t3); " in HLSL.

RegisterSpace

The register space. Can typically be 0, but allows multiple descriptor arrays of unknown size to not appear to overlap. For example, for SRVs, by extending the example in the `BaseShaderRegister` member description, 5 maps to ": register(t3,space5); " in HLSL.

Flags

Specifies the [D3D12_DESCRIPTOR_RANGE_FLAGS](#) that determine descriptor and data volatility.

OffsetInDescriptorsFromTableStart

The offset in descriptors from the start of the root signature. This value can be [D3D12_DESCRIPTOR_RANGE_OFFSET_APPEND](#), which indicates this range should immediately follow the preceding range.

Remarks

This structure is a member of the [D3D12_ROOT_DESCRIPTOR_TABLE1](#) structure.

Refer to the helper structure [CD3DX12_DESCRIPTOR_RANGE1](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_DEVICE_REMOVED_EXTENDED_DATA structure (d3d12.h)

Article02/22/2024

ⓘ Note

As of Windows 10, version 1903, `D3D12_DEVICE_REMOVED_EXTENDED_DATA` is deprecated, and it may not be available in future versions of Windows. Use [`D3D12 DEVICE REMOVED EXTENDED DATA1`](#) instead.

Represents Device Removed Extended Data (DRED) version 1.0 data.

Syntax

C++

```
typedef struct D3D12_DEVICE_REMOVED_EXTENDED_DATA {
    D3D12_DRED_FLAGS          Flags;
    D3D12_AUTO_BREADCRUMB_NODE *pHeadAutoBreadcrumbNode;
} D3D12_DEVICE_REMOVED_EXTENDED_DATA;
```

Members

Flags

An input parameter of type [`D3D12_DRED_FLAGS`](#), specifying control flags for the Direct3D runtime.

pHeadAutoBreadcrumbNode

An output parameter of type pointer to [`D3D12_AUTO_BREADCRUMB_NODE`](#) representing the returned auto-breadcrumb object(s). This is a pointer to the head of a linked list of auto-breadcrumb objects. All of the nodes in the linked list represent potentially incomplete command list execution on the GPU at the time of the device-removal event.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

[!\[\]\(73a97b58cbe5aa2e59d625ac5112460b_img.jpg\) Yes](#)[!\[\]\(5edfeeafeb1429ef05bac2eeb33dc7dc_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEVICE_REMOVED_EXTENDED_DATA1 structure (d3d12.h)

Article 02/22/2024

Represents Device Removed Extended Data (DRED) version 1.1 device removal data, so that debuggers and debugger extensions can access DRED data. Also see [D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA](#).

This structure is not used by any interface methods, and it provides no runtime API access.

Syntax

C++

```
typedef struct D3D12_DEVICE_REMOVED_EXTENDED_DATA1 {
    HRESULT DeviceRemovedReason;
    D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT AutoBreadcrumbsOutput;
    D3D12_DRED_PAGE_FAULT_OUTPUT PageFaultOutput;
} D3D12_DEVICE_REMOVED_EXTENDED_DATA1;
```

Members

`DeviceRemovedReason`

An [HRESULT](#) containing the reason the device was removed (matches the return value of [GetDeviceRemovedReason](#)). Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

`AutoBreadcrumbsOutput`

A [D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT](#) value that contains the auto-breadcrumb state prior to device removal.

`PageFaultOutput`

A [D3D12_DRED_PAGE_FAULT_OUTPUT](#) value that contains page fault data if device removal was the result of a GPU page fault.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 [Yes](#) [No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_DEVICE_REMOVED_EXTENDED_DATA2 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DEVICE_REMOVED_EXTENDED_DATA2 {
    HRESULT DeviceRemovedReason;
    D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1 AutoBreadcrumbsOutput;
    D3D12_DRED_PAGE_FAULT_OUTPUT1 PageFaultOutput;
} D3D12_DEVICE_REMOVED_EXTENDED_DATA2;
```

Members

DeviceRemovedReason

AutoBreadcrumbsOutput

PageFaultOutput

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEVICE_REMOVED_EXTENDED_DATA3 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DEVICE_REMOVED_EXTENDED_DATA3 {
    HRESULT DeviceRemovedReason;
    D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1 AutoBreadcrumbsOutput;
    D3D12_DRED_PAGE_FAULT_OUTPUT2 PageFaultOutput;
    D3D12_DRED_DEVICE_STATE DeviceState;
} D3D12_DEVICE_REMOVED_EXTENDED_DATA3;
```

Members

DeviceRemovedReason

AutoBreadcrumbsOutput

PageFaultOutput

DeviceState

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_DISCARD_REGION structure (d3d12.h)

Article02/22/2024

Describes details for the discard-resource operation.

Syntax

C++

```
typedef struct D3D12_DISCARD_REGION {
    UINT           NumRects;
    const D3D12_RECT *pRects;
    UINT           FirstSubresource;
    UINT           NumSubresources;
} D3D12_DISCARD_REGION;
```

Members

NumRects

The number of rectangles in the array that the **pRects** member specifies.

pRects

An array of **D3D12_RECT** structures for the rectangles in the resource to discard. If **NULL**, [DiscardResource](#) discards the entire resource.

FirstSubresource

Index of the first subresource in the resource to discard.

NumSubresources

The number of subresources in the resource to discard.

Remarks

This structure is used by the [ID3D12GraphicsCommandList::DiscardResource](#) method.

If rectangles are supplied in this structure, the resource must have 2D subresources with all specified subresources the same dimension.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DISPATCH_ARGUMENTS structure (d3d12.h)

Article 10/06/2021

Describes dispatch parameters, for use by the compute shader.

Syntax

C++

```
typedef struct D3D12_DISPATCH_ARGUMENTS {
    UINT ThreadGroupCountX;
    UINT ThreadGroupCountY;
    UINT ThreadGroupCountZ;
} D3D12_DISPATCH_ARGUMENTS;
```

Members

ThreadGroupCountX

The size, in thread groups, of the x-dimension of the thread-group grid.

ThreadGroupCountY

The size, in thread groups, of the y-dimension of the thread-group grid.

ThreadGroupCountZ

The size, in thread groups, of the z-dimension of the thread-group grid.

Remarks

The members of this structure serve the same purpose as the parameters of [Dispatch](#).

A compiled compute shader defines the set of instructions to execute per thread and the number of threads to run per group. The thread-group parameters indicate how many thread groups to execute. Each thread group contains the same number of threads, as defined by the compiled compute shader. The thread groups are organized in a three-dimensional grid. The total number of thread groups that the compiled compute shader executes is determined by the following calculation:

Syntax

```
ThreadGroupCountX * ThreadGroupCountY * ThreadGroupCountZ
```

In particular, if any of the values in the thread-group parameters are 0, nothing will happen.

The maximum size of any dimension is 65535.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DISPATCH_MESH_ARGUMENTS structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DISPATCH_MESH_ARGUMENTS {
    UINT ThreadGroupCountX;
    UINT ThreadGroupCountY;
    UINT ThreadGroupCountZ;
} D3D12_DISPATCH_MESH_ARGUMENTS;
```

Members

ThreadGroupCountX

ThreadGroupCountY

ThreadGroupCountZ

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DISPATCH_RAYS_DESC structure (d3d12.h)

Article04/02/2021

Describes the properties of a ray dispatch operation initiated with a call to [ID3D12GraphicsCommandList4::DispatchRays](#).

Syntax

C++

```
typedef struct D3D12_DISPATCH_RAYS_DESC {
    D3D12_GPU_VIRTUAL_ADDRESS_RANGE           RayGenerationShaderRecord;
    D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE MissShaderTable;
    D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE HitGroupTable;
    D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE CallableShaderTable;
    UINT                                     Width;
    UINT                                     Height;
    UINT                                     Depth;
} D3D12_DISPATCH_RAYS_DESC;
```

Members

RayGenerationShaderRecord

The shader record for the ray generation shader to use.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

The address must be aligned to 64 bytes, defined as [D3D12_RAYTRACING_SHADER_TABLE_BYTE_ALIGNMENT](#), and in the range [0...4096] bytes.

MissShaderTable

The shader table for miss shaders.

The stride is record stride, and must be aligned to 32 bytes, defined as [D3D12_RAYTRACING_SHADER_RECORD_BYTE_ALIGNMENT](#), and in the range [0...4096] bytes. 0 is allowed.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

The address must be aligned to 64 bytes, defined as [D3D12_RAYTRACING_SHADER_TABLE_BYTE_ALIGNMENT](#).

HitGroupTable

The shader table for hit groups.

The stride is record stride, and must be aligned to 32 bytes, defined as [D3D12_RAYTRACING_SHADER_RECORD_BYTE_ALIGNMENT](#), and in the range [0...4096] bytes. 0 is allowed.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

The address must be aligned to 64 bytes, defined as [D3D12_RAYTRACING_SHADER_TABLE_BYTE_ALIGNMENT](#).

CallableShaderTable

The shader table for callable shaders.

The stride is record stride, and must be aligned to 32 bytes, defined as [D3D12_RAYTRACING_SHADER_RECORD_BYTE_ALIGNMENT](#). 0 is allowed.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

The address must be aligned to 64 bytes, defined as [D3D12_RAYTRACING_SHADER_TABLE_BYTE_ALIGNMENT](#).

Width

The width of the generation shader thread grid.

Height

The height of the generation shader thread grid.

Depth

The depth of the generation shader thread grid.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_DRAW_ARGUMENTS structure (d3d12.h)

Article 02/22/2024

Describes parameters for drawing instances.

Syntax

C++

```
typedef struct D3D12_DRAW_ARGUMENTS {
    UINT VertexCountPerInstance;
    UINT InstanceCount;
    UINT StartVertexLocation;
    UINT StartInstanceLocation;
} D3D12_DRAW_ARGUMENTS;
```

Members

`VertexCountPerInstance`

Specifies the number of vertices to draw, per instance.

`InstanceCount`

Specifies the number of instances.

`StartVertexLocation`

Specifies an index to the first vertex to start drawing from.

`StartInstanceLocation`

Specifies an index to the first instance to start drawing from.

Remarks

The members of this structure serve the same purpose as the parameters of [DrawInstanced](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRAW_INDEXED_ARGUMENTS structure (d3d12.h)

Article02/22/2024

Describes parameters for drawing indexed instances.

Syntax

C++

```
typedef struct D3D12_DRAW_INDEXED_ARGUMENTS {
    UINT IndexCountPerInstance;
    UINT InstanceCount;
    UINT StartIndexLocation;
    INT  BaseVertexLocation;
    UINT StartInstanceLocation;
} D3D12_DRAW_INDEXED_ARGUMENTS;
```

Members

`IndexCountPerInstance`

The number of indices read from the index buffer for each instance.

`InstanceCount`

The number of instances to draw.

`StartIndexLocation`

The location of the first index read by the GPU from the index buffer.

`BaseVertexLocation`

A value added to each index before reading a vertex from the vertex buffer.

`StartInstanceLocation`

A value added to each index before reading per-instance data from a vertex buffer.

Remarks

The members of this structure serve the same purpose as the parameters of [DrawIndexedInstanced](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_ALLOCATION_NODE structure (d3d12.h)

Article02/22/2024

Describes, as a node in a linked list, data about an allocation tracked by Device Removed Extended Data (DRED). This data includes the GPU VA allocation ranges, and an associated runtime object debug name and type. Each

D3D12_DRED_ALLOCATION_NODE object is singly linked to the next via its `pNext` member; except for the last node in the list, which has its `pNext` set to `nullptr`. A linked list structure is necessary because a runtime object can share allocation ranges with other objects.

If device removal is caused by a GPU page fault—and DRED page fault reporting is enabled—then DRED builds a list of D3D12_DRED_ALLOCATION_NODE structs that includes all matching allocation nodes for active and recently-freed runtime objects.

Syntax

C++

```
typedef struct D3D12_DRED_ALLOCATION_NODE {
    const char                      *ObjectNameA;
    const wchar_t                    *ObjectNameW;
    D3D12_DRED_ALLOCATION_TYPE     AllocationType;
    const D3D12_DRED_ALLOCATION_NODE *pNext;
    struct                           D3D12_DRED_ALLOCATION_NODE;
} D3D12_DRED_ALLOCATION_NODE;
```

Members

`ObjectNameA`

A pointer to the ANSI debug name of the allocated runtime object.

`ObjectNameW`

A pointer to the wide debug name of the allocated runtime object.

`AllocationType`

A `D3D12_DRED_ALLOCATION_TYPE` value representing the runtime object's allocation type.

`pNext`

A pointer to a constant `D3D12_DRED_ALLOCATION_NODE` representing the next allocation node in the list, or `nullptr` if this is the last node.

`D3D12_DRED_ALLOCATION_NODE`

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_ALLOCATION_NODE1 structure (d3d12.h)

Article 02/22/2024

Syntax

C++

```
typedef struct D3D12_DRED_ALLOCATION_NODE1 {
    const char                      *ObjectNameA;
    const wchar_t                    *ObjectNameW;
    D3D12_DRED_ALLOCATION_TYPE     AllocationType;
    const D3D12_DRED_ALLOCATION_NODE1 *pNext;
    struct                           D3D12_DRED_ALLOCATION_NODE1;
    const IUnknown                   *pObject;
} D3D12_DRED_ALLOCATION_NODE1;
```

Members

ObjectNameA

ObjectNameW

AllocationType

pNext

D3D12_DRED_ALLOCATION_NODE1

pObject

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_ALLOCATION_TYPE enumeration (d3d12.h)

Article10/20/2021

Congruent with, and numerically equivalent to, [3D12DDI_HANDLETYPE](#) enumeration values.

Syntax

C++

```
typedef enum D3D12_DRED_ALLOCATION_TYPE {
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_QUEUE,
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_ALLOCATOR,
    D3D12_DRED_ALLOCATION_TYPE_PIPELINE_STATE,
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_LIST,
    D3D12_DRED_ALLOCATION_TYPE_FENCE,
    D3D12_DRED_ALLOCATION_TYPE_DESCRIPTOR_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_QUERY_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_SIGNATURE,
    D3D12_DRED_ALLOCATION_TYPE_PIPELINE_LIBRARY,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_DECODER,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_PROCESSOR,
    D3D12_DRED_ALLOCATION_TYPE_RESOURCE,
    D3D12_DRED_ALLOCATION_TYPE_PASS,
    D3D12_DRED_ALLOCATION_TYPE_CRYPTOSESSION,
    D3D12_DRED_ALLOCATION_TYPE_CRYPTOSESSIONPOLICY,
    D3D12_DRED_ALLOCATION_TYPE_PROTECTEDRESOURCESESSION,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_DECODER_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_POOL,
    D3D12_DRED_ALLOCATION_TYPE_COMMAND_RECORDER,
    D3D12_DRED_ALLOCATION_TYPE_STATE_OBJECT,
    D3D12_DRED_ALLOCATION_TYPE_METACOMMAND,
    D3D12_DRED_ALLOCATION_TYPE_SCHEDULINGGROUP,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_MOTION_ESTIMATOR,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_MOTION_VECTOR_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_EXTENSION_COMMAND,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_ENCODER,
    D3D12_DRED_ALLOCATION_TYPE_VIDEO_ENCODER_HEAP,
    D3D12_DRED_ALLOCATION_TYPE_INVALID
};
```

Constants

[Expand table](#)

D3D12_DRED_ALLOCATION_TYPE_COMMAND_QUEUE
Value: (19)
D3D12_DRED_ALLOCATION_TYPE_COMMAND_ALLOCATOR
Value: (20)
D3D12_DRED_ALLOCATION_TYPE_PIPELINE_STATE
Value: (21)
D3D12_DRED_ALLOCATION_TYPE_COMMAND_LIST
Value: (22)
D3D12_DRED_ALLOCATION_TYPE_FENCE
Value: (23)
D3D12_DRED_ALLOCATION_TYPE_DESCRIPTOR_HEAP
Value: (24)
D3D12_DRED_ALLOCATION_TYPE_HEAP
Value: (25)
D3D12_DRED_ALLOCATION_TYPE_QUERY_HEAP
Value: (27)
D3D12_DRED_ALLOCATION_TYPE_COMMAND_SIGNATURE
Value: (28)
D3D12_DRED_ALLOCATION_TYPE_PIPELINE_LIBRARY
Value: (29)
D3D12_DRED_ALLOCATION_TYPE_VIDEO_DECODER
Value: (30)
D3D12_DRED_ALLOCATION_TYPE_VIDEO_PROCESSOR
Value: (32)
D3D12_DRED_ALLOCATION_TYPE_RESOURCE
Value: (34)
D3D12_DRED_ALLOCATION_TYPE_PASS
Value: (35)
D3D12_DRED_ALLOCATION_TYPE_CRYPTOSESSION
Value: (36)
D3D12_DRED_ALLOCATION_TYPE_CRYPTOSESSIONPOLICY
Value: (37)

`D3D12_DRED_ALLOCATION_TYPE_PROTECTEDRESOURCESESSION`

Value: (38)

`D3D12_DRED_ALLOCATION_TYPE_VIDEO_DECODER_HEAP`

Value: (39)

`D3D12_DRED_ALLOCATION_TYPE_COMMAND_POOL`

Value: (40)

`D3D12_DRED_ALLOCATION_TYPE_COMMAND_RECORDER`

Value: (41)

`D3D12_DRED_ALLOCATION_TYPE_STATE_OBJECT`

Value: (42)

`D3D12_DRED_ALLOCATION_TYPE_METACOMMAND`

Value: (43)

`D3D12_DRED_ALLOCATION_TYPE_SCHEDULINGGROUP`

Value: (44)

`D3D12_DRED_ALLOCATION_TYPE_VIDEO_MOTION_ESTIMATOR`

Value: (45)

`D3D12_DRED_ALLOCATION_TYPE_VIDEO_MOTION_VECTOR_HEAP`

Value: (46)

`D3D12_DRED_ALLOCATION_TYPE_VIDEO_EXTENSION_COMMAND`

Value: (47)

`D3D12_DRED_ALLOCATION_TYPE_INVALID`

Value: (0xffffffff)

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- Core enumerations
 - Use DRED to diagnose GPU faults
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT structure (d3d12.h)

Article02/22/2024

Contains a pointer to the head of a linked list of [D3D12_AUTO_BREADCRUMB_NODE](#) objects. The list represents the auto-breadcrumb state prior to device removal.

Syntax

C++

```
typedef struct D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT {
    const D3D12_AUTO_BREADCRUMB_NODE *pHeadAutoBreadcrumbNode;
} D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT;
```

Members

pHeadAutoBreadcrumbNode

A pointer to a constant [D3D12_AUTO_BREADCRUMB_NODE](#) object representing the head of a linked list of auto-breadcrumb nodes, or `nullptr` if the list is empty.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1 {
    const D3D12_AUTO_BREADCRUMB_NODE1 *pHeadAutoBreadcrumbNode;
} D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1;
```

Members

pHeadAutoBreadcrumbNode

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_BREADCRUMB_CONTEXT structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DRED_BREADCRUMB_CONTEXT {
    UINT          BreadcrumbIndex;
    const wchar_t *pContextString;
} D3D12_DRED_BREADCRUMB_CONTEXT;
```

Members

BreadcrumbIndex

pContextString

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_DEVICE_STATE enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_DRED_DEVICE_STATE {
    D3D12_DRED_DEVICE_STATE_UNKNOWN,
    D3D12_DRED_DEVICE_STATE_HUNG,
    D3D12_DRED_DEVICE_STATE_FAULT,
    D3D12_DRED_DEVICE_STATE_PAGEFAULT
};
```

Constants

[+] Expand table

D3D12_DRED_DEVICE_STATE_UNKNOWN
D3D12_DRED_DEVICE_STATE_HUNG
D3D12_DRED_DEVICE_STATE_FAULT
D3D12_DRED_DEVICE_STATE_PAGEFAULT

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_DRED_ENABLEMENT enumeration (d3d12.h)

Article02/22/2024

Defines constants (used by the [ID3D12DeviceRemovedExtendedDataSettings interface](#)) that specify how individual Device Removed Extended Data (DRED) features are enabled. As of DRED version 1.1, the default value for all settings is **D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED**.

Syntax

C++

```
typedef enum D3D12_DRED_ENABLEMENT {
    D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED,
    D3D12_DRED_ENABLEMENT_FORCED_OFF,
    D3D12_DRED_ENABLEMENT_FORCED_ON
} ;
```

Constants

[] [Expand table](#)

D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED

Value: (0)

Specifies that a DRED feature is enabled only when DRED is turned on by the system automatically (for example, when a user is reproducing a problem via FeedbackHub).

D3D12_DRED_ENABLEMENT_FORCED_OFF

Value: (1)

Specifies that a DRED feature should be force-disabled, regardless of the system state.

D3D12_DRED_ENABLEMENT_FORCED_ON

Value: (2)

Specifies that a DRED feature should be force-enabled, regardless of the system state.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core enumerations](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

[!\[\]\(ba25252bfcad5d7dbf735955ded35139_img.jpg\) Yes](#)[!\[\]\(5c5e96fa5479e45b1a8819251fd28fb6_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_FLAGS enumeration (d3d12.h)

Article02/22/2024

ⓘ Note

As of Windows 10, version 1903, D3D12_DRED_FLAGS is deprecated, and it may not be available in future versions of Windows.

Defines constants used in the [D3D12_DEVICE_REMOVED_EXTENDED_DATA structure](#) to specify control flags for the Direct3D runtime. Values can be bitwise OR'd together.

Syntax

C++

```
typedef enum D3D12_DRED_FLAGS {
    D3D12_DRED_FLAG_NONE,
    D3D12_DRED_FLAG_FORCE_ENABLE,
    D3D12_DRED_FLAG_DISABLE_AUTOBREADCRUMBS
} ;
```

Constants

[+] [Expand table](#)

`D3D12_DRED_FLAG_NONE`

Value: `(0x0)`

Typically specifies that Device Removed Extended Data (DRED) is disabled, except for when user-initiated feedback is used to produce a repro, or when otherwise enabled by Windows via automatic detection of process-instability issues. This is the default value.

`D3D12_DRED_FLAG_FORCE_ENABLE`

Value: `(0x1)`

Forces DRED to be enabled, regardless of the system state.

`D3D12_DRED_FLAG_DISABLE_AUTOBREADCRUMBS`

Value: `(0x2)`

Disables DRED auto breadcrumbs.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core enumerations](#)
- [D3D12_DEVICE_REMOVED_EXTENDED_DATA structure](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_PAGE_FAULT_FLAGS enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_DRED_PAGE_FAULT_FLAGS {
    D3D12_DRED_PAGE_FAULT_FLAGS_NONE
} ;
```

Constants

[] Expand table

D3D12_DRED_PAGE_FAULT_FLAGS_NONE

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_PAGE_FAULT_OUTPUT structure (d3d12.h)

Article02/22/2024

Describes allocation data related to a GPU page fault on a given virtual address (VA). Contains the VA of a GPU page fault, together with a list of matching allocation nodes for active objects, and a list of allocation nodes for recently deleted objects.

Syntax

C++

```
typedef struct D3D12_DRED_PAGE_FAULT_OUTPUT {
    D3D12_GPU_VIRTUAL_ADDRESS           PageFaultVA;
    const D3D12_DRED_ALLOCATION_NODE *pHeadExistingAllocationNode;
    const D3D12_DRED_ALLOCATION_NODE *pHeadRecentFreedAllocationNode;
} D3D12_DRED_PAGE_FAULT_OUTPUT;
```

Members

`PageFaultVA`

A [D3D12_GPU_VIRTUAL_ADDRESS](#) containing the GPU virtual address (VA) of the faulting operation if device removal was due to a GPU page fault.

`pHeadExistingAllocationNode`

A pointer to a constant [D3D12_DRED_ALLOCATION_NODE](#) object representing the head of a linked list of allocation nodes for active allocated runtime objects with virtual address (VA) ranges that match the faulting VA (`PageFaultVA`). Has a value of `nullptr` if the list is empty.

`pHeadRecentFreedAllocationNode`

A pointer to a constant [D3D12_DRED_ALLOCATION_NODE](#) object representing the head of a linked list of allocation nodes for recently freed runtime objects with virtual address (VA) ranges that match the faulting VA (`PageFaultVA`). Has a value of `nullptr` if the list is empty.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

[!\[\]\(566ca5ab425b1076d2ff8e3ef6e03559_img.jpg\) Yes](#)[!\[\]\(575caef3bc52c5551caf29f47f4faefe_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_PAGE_FAULT_OUTPUT1 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DRED_PAGE_FAULT_OUTPUT1 {
    D3D12_GPU_VIRTUAL_ADDRESS          PageFaultVA;
    const D3D12_DRED_ALLOCATION_NODE1 *pHeadExistingAllocationNode;
    const D3D12_DRED_ALLOCATION_NODE1 *pHeadRecentFreedAllocationNode;
} D3D12_DRED_PAGE_FAULT_OUTPUT1;
```

Members

PageFaultVA

pHeadExistingAllocationNode

pHeadRecentFreedAllocationNode

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRED_PAGE_FAULT_OUTPUT2 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_DRED_PAGE_FAULT_OUTPUT2 {
    D3D12_GPU_VIRTUAL_ADDRESS          PageFaultVA;
    const D3D12_DRED_ALLOCATION_NODE1 *pHeadExistingAllocationNode;
    const D3D12_DRED_ALLOCATION_NODE1 *pHeadRecentFreedAllocationNode;
    D3D12_DRED_PAGE_FAULT_FLAGS       PageFaultFlags;
} D3D12_DRED_PAGE_FAULT_OUTPUT2;
```

Members

PageFaultVA

pHeadExistingAllocationNode

pHeadRecentFreedAllocationNode

PageFaultFlags

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_DRED_VERSION enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a version of Device Removed Extended Data (DRED), as used by the [D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure](#).

Syntax

C++

```
typedef enum D3D12_DRED_VERSION {
    D3D12_DRED_VERSION_1_0,
    D3D12_DRED_VERSION_1_1,
    D3D12_DRED_VERSION_1_2,
    D3D12_DRED_VERSION_1_3
};
```

Constants

[] Expand table

<code>D3D12_DRED_VERSION_1_0</code>	Value: <code>(0x1)</code> Specifies DRED version 1.0.
<code>D3D12_DRED_VERSION_1_1</code>	Value: <code>(0x2)</code> Specifies DRED version 1.1.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h

See also

- [Core enumerations](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS enumeration (d3d12.h)

Article 02/22/2024

Specifies the result of a call to [ID3D12Device5::CheckDriverMatchingIdentifier](#) which queries whether serialized data is compatible with the current device and driver version.

Syntax

C++

```
typedef enum D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS {
    D3D12_DRIVER_MATCHING_IDENTIFIER_COMPATIBLE_WITH_DEVICE = 0,
    D3D12_DRIVER_MATCHING_IDENTIFIER_UNSUPPORTED_TYPE = 0x1,
    D3D12_DRIVER_MATCHING_IDENTIFIER_UNRECOGNIZED = 0x2,
    D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_VERSION = 0x3,
    D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_TYPE = 0x4
};
```

Constants

[+] [Expand table](#)

<code>D3D12_DRIVER_MATCHING_IDENTIFIER_COMPATIBLE_WITH_DEVICE</code>
--

Value: 0

Serialized data is compatible with the current device/driver.

<code>D3D12_DRIVER_MATCHING_IDENTIFIER_UNSUPPORTED_TYPE</code>
--

Value: 0x1

The specified [D3D12_SERIALIZED_DATA_TYPE](#) specified is unknown or unsupported.

<code>D3D12_DRIVER_MATCHING_IDENTIFIER_UNRECOGNIZED</code>
--

Value: 0x2

Format of the data in [D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER](#) is unrecognized. This could indicate either corrupt data or the identifier was produced by a different hardware vendor.

<code>D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_VERSION</code>
--

Value: 0x3

Serialized data is recognized, but its version is not compatible with the current driver. This result may indicate that the device is from the same hardware vendor but is an incompatible version.

`D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_TYPE`

Value: 0x4

`D3D12_SERIALIZED_DATA_TYPE` specifies a data type that is not compatible with the type of serialized data. As long as there is only a single defined serialized data type this error cannot not be produced.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DSV_DIMENSION enumeration (d3d12.h)

Article02/22/2024

Specifies how to access a resource used in a depth-stencil view.

Syntax

C++

```
typedef enum D3D12_DSV_DIMENSION {
    D3D12_DSV_DIMENSION_UNKNOWN = 0,
    D3D12_DSV_DIMENSION_TEXTURE1D = 1,
    D3D12_DSV_DIMENSION_TEXTURE1DARRAY = 2,
    D3D12_DSV_DIMENSION_TEXTURE2D = 3,
    D3D12_DSV_DIMENSION_TEXTURE2DARRAY = 4,
    D3D12_DSV_DIMENSION_TEXTURE2DMS = 5,
    D3D12_DSV_DIMENSION_TEXTURE2DMSARRAY = 6
};
```

Constants

[+] Expand table

D3D12_DSV_DIMENSION_UNKNOWN

Value: 0

D3D12_DSV_DIMENSION_UNKNOWN is not a valid value for [D3D12_DEPTH_STENCIL_VIEW_DESC](#) and is not used.

D3D12_DSV_DIMENSION_TEXTURE1D

Value: 1

The resource will be accessed as a 1D texture.

D3D12_DSV_DIMENSION_TEXTURE1DARRAY

Value: 2

The resource will be accessed as an array of 1D textures.

D3D12_DSV_DIMENSION_TEXTURE2D

Value: 3

The resource will be accessed as a 2D texture.

`D3D12_DSV_DIMENSION_TEXTURE2DARRAY`

Value: 4

The resource will be accessed as an array of 2D textures.

`D3D12_DSV_DIMENSION_TEXTURE2DMS`

Value: 5

The resource will be accessed as a 2D texture with multi sampling.

`D3D12_DSV_DIMENSION_TEXTURE2DMSARRAY`

Value: 6

The resource will be accessed as an array of 2D textures with multi sampling.

Remarks

Specify one of the values in this enumeration in the `ViewDimension` member of a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_DSV_FLAGS enumeration (d3d12.h)

Article 02/22/2024

Specifies depth-stencil view options.

Syntax

C++

```
typedef enum D3D12_DSV_FLAGS {
    D3D12_DSV_FLAG_NONE = 0,
    D3D12_DSV_FLAG_READ_ONLY_DEPTH = 0x1,
    D3D12_DSV_FLAG_READ_ONLY_STENCIL = 0x2
} ;
```

Constants

[+] Expand table

<code>D3D12_DSV_FLAG_NONE</code>	Value: <i>0</i> Indicates a default view.
<code>D3D12_DSV_FLAG_READ_ONLY_DEPTH</code>	Value: <i>0x1</i> Indicates that depth values are read only.
<code>D3D12_DSV_FLAG_READ_ONLY_STENCIL</code>	Value: <i>0x2</i> Indicates that stencil values are read only.

Remarks

Specify a combination of the values in this enumeration in the **Flags** member of a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure. The values are combined by using a bitwise OR operation.

Limiting a depth-stencil buffer to read-only access allows more than one depth-stencil view to be bound to the pipeline simultaneously, since it is not possible to have read/write conflicts between separate views.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DXIL_LIBRARY_DESC structure (d3d12.h)

Article 02/22/2024

Describes a DXIL library state subobject that can be included in a state object.

Syntax

C++

```
typedef struct D3D12_DXIL_LIBRARY_DESC {
    D3D12_SHADER_BYTECODE    DXILLibrary;
    UINT                     NumExports;
    const D3D12_EXPORT_DESC *pExports;
} D3D12_DXIL_LIBRARY_DESC;
```

Members

DXILLibrary

The library to include in the state object. Must have been compiled with library target 6.3 or higher. It is fine to specify the same library multiple times either in the same state object / collection or across multiple, as long as the names exported each time don't conflict in a given state object.

NumExports

The size of *pExports* array. If 0, everything gets exported from the library.

pExports

Optional exports array. For more information, see [D3D12_EXPORT_DESC](#).

pExports

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure (d3d12.h)

Article02/22/2024

This subobject is unsupported in the current release.

Syntax

C++

```
typedef struct D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION {
    LPCWSTR SubobjectToAssociate;
    UINT     NumExports;
    LPCWSTR *pExports;
} D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION;
```

Members

SubobjectToAssociate

NumExports

Size of the *pExports* array. If 0, this is being explicitly defined as a default association. Another way to define a default association is to omit this subobject association for that subobject completely.

pExports

The array of exports with which the subobject is associated.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ELEMENTS_LAYOUT enumeration (d3d12.h)

Article 02/22/2024

Describes how the locations of elements are identified.

Syntax

C++

```
typedef enum D3D12_ELEMENTS_LAYOUT {
    D3D12_ELEMENTS_LAYOUT_ARRAY = 0,
    D3D12_ELEMENTS_LAYOUT_ARRAY_OF_POINTERS = 0x1
};
```

Constants

 Expand table

D3D12_ELEMENTS_LAYOUT_ARRAY Value: 0 For a data set of n elements, the pointer parameter points to the start of n elements in memory.
D3D12_ELEMENTS_LAYOUT_ARRAY_OF_POINTERS Value: 0x1 For a data set of n elements, the pointer parameter points to an array of n pointers in memory, each pointing to an individual element of the set.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ENCODE_ANISOTROPIC_FILTER macro (d3d12.h)

Article 02/22/2024

Syntax

C++

```
void D3D12_ENCODE_ANISOTROPIC_FILTER(  
    reduction  
);
```

Parameters

reduction

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗ | Get help at Microsoft Q&A

D3D12_ENCODE_BASIC_FILTER macro (d3d12.h)

Article 02/22/2024

Syntax

C++

```
void D3D12_ENCODE_BASIC_FILTER(
    min,
    mag,
    mip,
    reduction
);
```

Parameters

min

mag

mip

reduction

Return value

None

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING(
    Src0,
    Src1,
    Src2,
    Src3
);
```

Parameters

Src0

Src1

Src2

Src3

Return value

None

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_EXISTING_COLLECTION_DESC structure (d3d12.h)

Article02/22/2024

A state subobject describing an existing collection that can be included in a state object.

Syntax

C++

```
typedef struct D3D12_EXISTING_COLLECTION_DESC {
    ID3D12StateObject      *pExistingCollection;
    UINT                   NumExports;
    const D3D12_EXPORT_DESC *pExports;
} D3D12_EXISTING_COLLECTION_DESC;
```

Members

pExistingCollection

The collection to include in a state object. The enclosing state object holds a reference to the existing collection.

NumExports

Size of the *pExports* array. If 0, all of the collection's exports get exported.

pExports

Optional exports array. For more information, see [D3D12_EXPORT_DESC](#).

pExports

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_EXPORT_DESC structure (d3d12.h)

Article 02/22/2024

Describes an export from a state subobject such as a DXIL library or a collection state object.

Syntax

C++

```
typedef struct D3D12_EXPORT_DESC {
    LPCWSTR           Name;
    LPCWSTR           ExportToRename;
    D3D12_EXPORT_FLAGS Flags;
} D3D12_EXPORT_DESC;
```

Members

Name

The name to be exported. If the name refers to a function that is overloaded, a modified version of the name (e.g. encoding function parameter information in name string) can be provided to disambiguate which overload to use. The modified name for a function can be retrieved using HLSL compiler reflection.

If the *ExportToRename* field is non-null, *Name* refers to the new name to use for it when exported. In this case *Name* must be the unmodified name, whereas *ExportToRename* can be either a modified or unmodified name. A given internal name may be exported multiple times with different renames (and/or not renamed).

ExportToRename

If non-null, this is the name of an export to use but then rename when exported.

Flags

The flags to apply to the export.

Flags

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_EXPORT_FLAGS enumeration (d3d12.h)

Article 02/22/2024

The flags to apply when exporting symbols from a state subobject.

Syntax

C++

```
typedef enum D3D12_EXPORT_FLAGS {
    D3D12_EXPORT_FLAG_NONE = 0
} ;
```

Constants

[] Expand table

D3D12_EXPORT_FLAG_NONE

Value: 0

No export flags.

Remarks

No export flags are defined in the current release.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE enumeration (d3d12.h)

Article 02/14/2023

Defines constants that specify a Direct3D 12 feature or feature set to query about. When you want to query for the level to which an adapter supports a feature, pass one of these values to [ID3D12Device::CheckFeatureSupport](#).

Syntax

C++

```
typedef enum D3D12_FEATURE {
    D3D12_FEATURE_D3D12_OPTIONS = 0,
    D3D12_FEATURE_ARCHITECTURE = 1,
    D3D12_FEATURE_FEATURE_LEVELS = 2,
    D3D12_FEATURE_FORMAT_SUPPORT = 3,
    D3D12_FEATURE_MULTISAMPLE_QUALITY_LEVELS = 4,
    D3D12_FEATURE_FORMAT_INFO = 5,
    D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT = 6,
    D3D12_FEATURE_SHADER_MODEL = 7,
    D3D12_FEATURE_D3D12_OPTIONS1 = 8,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_SUPPORT = 10,
    D3D12_FEATURE_ROOT_SIGNATURE = 12,
    D3D12_FEATURE_ARCHITECTURE1 = 16,
    D3D12_FEATURE_D3D12_OPTIONS2 = 18,
    D3D12_FEATURE_SHADER_CACHE = 19,
    D3D12_FEATURE_COMMAND_QUEUE_PRIORITY = 20,
    D3D12_FEATURE_D3D12_OPTIONS3 = 21,
    D3D12_FEATURE_EXISTING_HEAPS = 22,
    D3D12_FEATURE_D3D12_OPTIONS4 = 23,
    D3D12_FEATURE_SERIALIZATION = 24,
    D3D12_FEATURE_CROSS_NODE = 25,
    D3D12_FEATURE_D3D12_OPTIONS5 = 27,
    D3D12_FEATURE_DISPLAYABLE,
    D3D12_FEATURE_D3D12_OPTIONS6 = 30,
    D3D12_FEATURE_QUERY_META_COMMAND = 31,
    D3D12_FEATURE_D3D12_OPTIONS7 = 32,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPE_COUNT = 33,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPES = 34,
    D3D12_FEATURE_D3D12_OPTIONS8 = 36,
    D3D12_FEATURE_D3D12_OPTIONS9 = 37,
    D3D12_FEATURE_D3D12_OPTIONS10,
    D3D12_FEATURE_D3D12_OPTIONS11,
    D3D12_FEATURE_D3D12_OPTIONS12,
    D3D12_FEATURE_D3D12_OPTIONS13,
    D3D12_FEATURE_D3D12_OPTIONS14,
    D3D12_FEATURE_D3D12_OPTIONS15,
    D3D12_FEATURE_D3D12_OPTIONS16,
    D3D12_FEATURE_D3D12_OPTIONS17,
```

```
D3D12_FEATURE_D3D12_OPTIONS18,  
D3D12_FEATURE_D3D12_OPTIONS19,  
D3D12_FEATURE_D3D12_OPTIONS20,  
D3D12_FEATURE_PREDICATION,  
D3D12_FEATURE_PLACED_RESOURCE_SUPPORT_INFO,  
D3D12_FEATURE_HARDWARE_COPY,  
D3D12_FEATURE_D3D12_OPTIONS21  
} ;
```

Constants

[+] Expand table

D3D12_FEATURE_D3D12_OPTIONS

Value: 0

Indicates a query for the level of support for basic Direct3D 12 feature options. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS](#).

D3D12_FEATURE_ARCHITECTURE

Value: 1

Indicates a query for the adapter's architectural details, so that your application can better optimize for certain adapter properties. The corresponding data structure for this value is [D3D12_FEATURE_DATA_ARCHITECTURE](#).

Note This value has been superseded by the [D3D_FEATURE_DATA_ARCHITECTURE1](#) value. If your application targets Windows 10, version 1703 (Creators' Update) or higher, then use the [D3D_FEATURE_DATA_ARCHITECTURE1](#) value instead.

D3D12_FEATURE_FEATURE_LEVELS

Value: 2

Indicates a query for info about the [feature levels](#) supported. The corresponding data structure for this value is [D3D12_FEATURE_DATA_FEATURE_LEVELS](#).

D3D12_FEATURE_FORMAT_SUPPORT

Value: 3

Indicates a query for the resources supported by the current graphics driver for a given format. The corresponding data structure for this value is [D3D12_FEATURE_DATA_FORMAT_SUPPORT](#).

D3D12_FEATURE_MULTISAMPLE_QUALITY_LEVELS

Value: 4

Indicates a query for the image quality levels for a given format and sample count. The corresponding data structure for this value is [D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS](#).

D3D12_FEATURE_FORMAT_INFO

Value: 5

Indicates a query for the DXGI data format. The corresponding data structure for this value is [D3D12_FEATURE_DATA_FORMAT_INFO](#).

D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT

Value: 6

Indicates a query for the GPU's virtual address space limitations. The corresponding data structure for this value is [D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT](#).

D3D12_FEATURE_SHADER_MODEL

Value: 7

Indicates a query for the supported shader model. The corresponding data structure for this value is [D3D12_FEATURE_DATA_SHADER_MODEL](#).

D3D12_FEATURE_D3D12_OPTIONS1

Value: 8

Indicates a query for the level of support for HLSL 6.0 wave operations. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS1](#).

D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_SUPPORT

Value: 10

Indicates a query for the level of support for protected resource sessions. The corresponding data structure for this value is [D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT](#).

D3D12_FEATURE_ROOT_SIGNATURE

Value: 12

Indicates a query for root signature version support. The corresponding data structure for this value is [D3D12_FEATURE_DATA_ROOT_SIGNATURE](#).

D3D12_FEATURE_ARCHITECTURE1

Value: 16

Indicates a query for each adapter's architectural details, so that your application can better optimize for certain adapter properties. The corresponding data structure for this value is [D3D12_FEATURE_DATA_ARCHITECTURE1](#).

Note This value supersedes the `D3D_FEATURE_DATA_ARCHITECTURE` value. If your application targets Windows 10, version 1703 (Creators' Update) or higher, then use [D3D_FEATURE_DATA_ARCHITECTURE1](#).

`D3D12_FEATURE_D3D12_OPTIONS2`

Value: 18

Indicates a query for the level of support for depth-bounds tests and programmable sample positions. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_D3D12_OPTIONS2](#).

`D3D12_FEATURE_SHADER_CACHE`

Value: 19

Indicates a query for the level of support for shader caching. The corresponding data structure for this value is [D3D12_FEATURE_DATA_SHADER_CACHE](#).

`D3D12_FEATURE_COMMAND_QUEUE_PRIORITY`

Value: 20

Indicates a query for the adapter's support for prioritization of different command queue types.

The corresponding data structure for this value is

[D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY](#).

`D3D12_FEATURE_D3D12_OPTIONS3`

Value: 21

Indicates a query for the level of support for timestamp queries, format-casting, immediate write, view instancing, and barycentrics. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_D3D12_OPTIONS3](#).

`D3D12_FEATURE_EXISTING_HEAPS`

Value: 22

Indicates a query for whether or not the adapter supports creating heaps from existing system memory. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_EXISTING_HEAPS](#).

`D3D12_FEATURE_D3D12_OPTIONS4`

Value: 23

Indicates a query for the level of support for 64KB-aligned MSAA textures, cross-API sharing, and native 16-bit shader operations. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_D3D12_OPTIONS4](#).

`D3D12_FEATURE_SERIALIZATION`

Value: 24

Indicates a query for the level of support for heap serialization. The corresponding data structure for this value is [D3D12_FEATURE_DATA_SERIALIZATION](#).

`D3D12_FEATURE_CROSS_NODE`

Value: 25

Indicates a query for the level of support for the sharing of resources between different adapters—for example, multiple GPUs. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_CROSS_NODE](#).

`D3D12_FEATURE_D3D12_OPTIONS5`

Value: 27

Starting with Windows 10, version 1809 (10.0; Build 17763), indicates a query for the level of support for render passes, ray tracing, and shader-resource view tier 3 tiled resources. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#).

D3D12_FEATURE_DISPLAYABLE

Starting with Windows 11 (Build 10.0.22000.194). The corresponding data structure for this value is [D3D12_FEATURE_DATA_DISPLAYABLE](#).

D3D12_FEATURE_D3D12_OPTIONS6

Value: 30

Starting with Windows 10, version 1903 (10.0; Build 18362), indicates a query for the level of support for variable-rate shading (VRS), and indicates whether or not background processing is supported. The corresponding data structure for this value is

[D3D12_FEATURE_DATA_D3D12_OPTIONS6](#).

For more info, see [Variable-rate shading \(VRS\)](#), and the [Direct3D 12 background processing spec](#).

D3D12_FEATURE_QUERY_META_COMMAND

Value: 31

Indicates a query for the level of support for metacommands. The corresponding data structure for this value is [D3D12_FEATURE_DATA_QUERY_META_COMMAND](#).

D3D12_FEATURE_D3D12_OPTIONS7

Value: 32

Starting with Windows 10, version 2004 (10.0; Build 19041), indicates a query for the level of support for mesh and amplification shaders, and for sampler feedback. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS7](#).

For more info, see the [Mesh shader](#) and [Sampler feedback](#) specs.

D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPE_COUNT

Value: 33

Starting with Windows 10, version 2004 (10.0; Build 19041), indicates a query to retrieve the count of protected resource session types. The corresponding data structure for this value is [D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT](#).

D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPES

Value: 34

Starting with Windows 10, version 2004 (10.0; Build 19041), indicates a query to retrieve the list of protected resource session types. The corresponding data structure for this value is [D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES](#).

D3D12_FEATURE_D3D12_OPTIONS8

Value: 36

Starting with Windows 11 (Build 10.0.22000.194), indicates whether or not unaligned block-compressed textures are supported. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS8](#).

D3D12_FEATURE_D3D12_OPTIONS9

Value: 37

Starting with Windows 11 (Build 10.0.22000.194), indicates whether or not support exists for mesh shaders, values of *SV_RenderTargetArrayIndex* that are 8 or greater, typed resource 64-bit integer atomics, derivative and derivative-dependent texture sample operations, and the level of support for WaveMMA (wave_matrix) operations. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS9](#).

D3D12_FEATURE_D3D12_OPTIONS10

Starting with Windows 11 (Build 10.0.22000.194), indicates whether or not the SUM combiner can be used, and whether or not *SV_ShadingRate* can be set from a mesh shader. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS10](#).

D3D12_FEATURE_D3D12_OPTIONS11

Starting with Windows 11 (Build 10.0.22000.194), indicates whether or not 64-bit integer atomics on resources in descriptor heaps are supported. The corresponding data structure for this value is [D3D12_FEATURE_DATA_D3D12_OPTIONS11](#).

Remarks

Use a constant from this enumeration in a call to [ID3D12Device::CheckFeatureSupport](#) to query a driver about support for various Direct3D 12 features. Each value in this enumeration has a corresponding data structure that you must pass (by pointer reference) in the *pFeatureSupportData* parameter of [ID3D12Device::CheckFeatureSupport](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

- [Core enumerations](#)
- [ID3D12Device::CheckFeatureSupport](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_ARCHITECTURE structure (d3d12.h)

Article04/02/2021

Provides detail about the adapter architecture, so that your application can better optimize for certain adapter properties.

Note This structure has been superseded by the [D3D12 FEATURE DATA ARCHITECTURE1](#) structure. If your application targets Windows 10, version 1703 (Creators' Update) or higher, then use [D3D12_FEATURE_DATA_ARCHITECTURE1](#) (and [D3D12 FEATURE ARCHITECTURE1](#)) instead.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_ARCHITECTURE {
    UINT NodeIndex;
    BOOL TileBasedRenderer;
    BOOL UMA;
    BOOL CacheCoherentUMA;
} D3D12_FEATURE_DATA_ARCHITECTURE;
```

Members

NodeIndex

In multi-adapter operation, this indicates which physical adapter of the device is relevant. See [Multi-adapter systems](#). **NodeIndex** is filled out by the application before calling [CheckFeatureSupport](#), as the application can retrieve details about the architecture of each adapter.

TileBasedRenderer

Specifies whether the hardware and driver support a tile-based renderer. The runtime sets this member to **TRUE** if the hardware and driver support a tile-based renderer.

UMA

Specifies whether the hardware and driver support UMA. The runtime sets this member to **TRUE** if the hardware and driver support UMA.

CacheCoherentUMA

Specifies whether the hardware and driver support cache-coherent UMA. The runtime sets this member to **TRUE** if the hardware and driver support cache-coherent UMA.

Remarks

How to use UMA and CacheCoherentUMA

D3D12 apps should be concerned about managing memory residency and providing the optimal heap properties. D3D12 apps can stay simplified and run reasonably well across many GPU architectures by only managing the residency for resources in `D3D12_HEAP_TYPE_DEFAULT` heaps. Those apps only need to call `IDXGIAdapter3::QueryVideoMemoryInfo` for `DXGI_MEMORY_SEGMENT_GROUP_LOCAL`, and they must be tolerant that `D3D12_HEAP_TYPE_UPLOAD` and `D3D12_HEAP_TYPE_READBACK` come from that same memory segment group.

However, such a simple design is too constraining for applications that push the limits. So, `D3D12_FEATURE_DATA_ARCHITECTURE` helps applications better optimize for the underlying adapter properties.

Some applications may want to better optimize for discrete adapters, and take on the additional complexity of managing both system memory and video memory budgets. If the size of upload heaps rivals the size of default textures, a near doubling of memory utilization is available. When supporting such optimizations, an application can either detect two residency budgets or recognize **UMA** is **false**.

Some applications may want to better optimize for integrated/ UMA adapters, especially those that are interested in extending battery life on mobile device. Simple D3D12 applications are forced into copying data between heaps with different attributions, when it isn't always necessary on UMA. However, the UMA property, by itself, encompasses a reasonably vague grey area of GPU designs. Do not assume UMA means all GPU-accessible memory can be freely made CPU-accessible, because it doesn't. There's a property that more closely aligns to that type of thinking:
`CacheCoherentUMA`.

When **CacheCoherentUMA** is **false**, a single residency budget is available but the UMA design commonly benefits from the three heap attributions. Opportunities do exist to remove resource copying through wise usage of upload and readback resources and heaps, that provide CPU-access to the memory. Such opportunities are not clear-cut, though. So, applications should be cautious; and experimentation across a variety of "UMA" systems is advisable, as resorting to enabling or precluding certain device IDs may be warranted. An understanding of the GPU memory architecture and how heap types translate to cache properties is recommended. The feasibility of success is likely dependent on how often each processor either reads or writes the data, the size and locality of data accesses, etc. For advanced developers: when **UMA** is true and **CacheCoherentUMA** is **false**, the most unique characteristic for these adapters is that upload heaps are still write-combined. However, some UMA adapters benefit from both the no-CPU-access and write-combine properties of default and upload heaps. See [GetCustomHeapProperties](#) for more details.

When **CacheCoherentUMA** is true, applications can more strongly entertain abandoning the attribution of heaps and using the custom heap equivalent of upload heaps everywhere. Zero-copy UMA optimizations are more generally encouraged as more scenarios will just benefit from shared usage. The memory model is very conducive to more scenarios and wider adoption. Some corner cases may still exist where benefits are not easily obtained, but they should be much rarer and less detrimental than other options. For advanced developers: **CacheCoherentUMA** means that a significant amount of the caches in the memory hierarchy are also unified or integrated between the CPU and GPU. The most unique observable characteristic is that upload heaps are actually write-back on **CacheCoherentUMA**. For these architecture, the usage of write-combine on upload heaps is commonly a detriment.

The low-level details should be ignored by a vast majority of single-adapter applications. As usual, single-adapter applications can simplify the landscape and ensure that the CPU writes to upload heaps use patterns that are write-combine-friendly. The lower-level details help reinforce the concepts for multi-adapter applications. Multi-adapter applications likely need to understand adapter architecture properties well enough to choose the optimal custom heap properties to efficiently move data between adapters.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_ARCHITECTURE1 structure (d3d12.h)

Article 08/19/2022

Provides detail about each adapter's architectural details, so that your application can better optimize for certain adapter properties.

Note This structure, introduced in Windows 10, version 1703 (Creators' Update), supersedes the [D3D12 FEATURE DATA ARCHITECTURE](#) structure. If your application targets Windows 10, version 1703 (Creators' Update) or higher, then use **D3D12_FEATURE_DATA_ARCHITECTURE1** (and [D3D12 FEATURE ARCHITECTURE1](#)).

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_ARCHITECTURE1 {
    UINT NodeIndex;
    BOOL TileBasedRenderer;
    BOOL UMA;
    BOOL CacheCoherentUMA;
    BOOL IsolatedMMU;
} D3D12_FEATURE_DATA_ARCHITECTURE1;
```

Members

NodeIndex

In multi-adapter operation, this indicates which physical adapter of the device is relevant. See [Multi-adapter systems](#). **NodeIndex** is filled out by the application before calling [CheckFeatureSupport](#), as the application can retrieve details about the architecture of each adapter.

TileBasedRenderer

Specifies whether the hardware and driver support a tile-based renderer. The runtime sets this member to **TRUE** if the hardware and driver support a tile-based renderer.

UMA

Specifies whether the hardware and driver support UMA. The runtime sets this member to **TRUE** if the hardware and driver support UMA.

CacheCoherentUMA

Specifies whether the hardware and driver support cache-coherent UMA. The runtime sets this member to **TRUE** if the hardware and driver support cache-coherent UMA.

IsolatedMMU

SAL: *out*

Specifies whether the hardware and driver support isolated Memory Management Unit (MMU). The runtime sets this member to **TRUE** if the GPU honors CPU page table properties like **MEM_WRITE_WATCH** (for more information, see [VirtualAlloc](#)) and **PAGE_READONLY** (for more information, see [Memory Protection Constants](#)).

If **TRUE**, the application must take care to no use memory with these page table properties with the GPU, as the GPU might trigger these page table properties in unexpected ways. For example, GPU write operations might be coarser than the application expects, particularly writes from within shaders. Certain write-watch pages might appear dirty, even when it isn't obvious how GPU writes may have affected them. GPU operations associated with upload and readback heap usage scenarios work well with write-watch pages, but might occasionally generate false positives that can be safely ignored.

Remarks

How to use UMA and CacheCoherentUMA

D3D12 apps should be concerned about managing memory residency and providing the optimal heap properties. D3D12 apps can stay simplified and run reasonably well across many GPU architectures by only managing the residency for resources in **D3D12_HEAP_TYPE_DEFAULT** heaps. Those apps only need to call [IDXGIAdapter3::QueryVideoMemoryInfo](#) for **DXGI_MEMORY_SEGMENT_GROUP_LOCAL**, and they must be tolerant that **D3D12_HEAP_TYPE_UPLOAD** and **D3D12_HEAP_TYPE_READBACK** come from that same memory segment group.

However, such a simple design is too constraining for applications that push the limits. So, **D3D12_FEATURE_DATA_ARCHITECTURE** helps applications better optimize for the

underlying adapter properties.

Some applications may want to better optimize for discrete adapters, and take on the additional complexity of managing both system memory and video memory budgets. If the size of upload heaps rivals the size of default textures, a near doubling of memory utilization is available. When supporting such optimizations, an application can either detect two residency budgets or recognize **UMA** is **false**.

Some applications may want to better optimize for integrated/ UMA adapters, especially those that are interested in extending battery life on mobile device. Simple D3D12 applications are forced into copying data between heaps with different attributions, when it isn't always necessary on UMA. However, the UMA property, by itself, encompasses a reasonably vague grey area of GPU designs. Do not assume UMA means all GPU-accessible memory can be freely made CPU-accessible, because it doesn't. There's a property that more closely aligns to that type of thinking:

CacheCoherentUMA.

When **CacheCoherentUMA** is **false**, a single residency budget is available but the UMA design commonly benefits from the three heap attributions. Opportunities do exist to remove resource copying through wise usage of upload and readback resources and heaps, that provide CPU-access to the memory. Such opportunities are not clear-cut, though. So, applications should be cautious; and experimentation across a variety of "UMA" systems is advisable, as resorting to enabling or precluding certain device IDs may be warranted. An understanding of the GPU memory architecture and how heap types translate to cache properties is recommended. The feasibility of success is likely dependent on how often each processor either reads or writes the data, the size and locality of data accesses, etc. For advanced developers: when **UMA** is true and **CacheCoherentUMA** is **false**, the most unique characteristic for these adapters is that upload heaps are still write-combined. However, some UMA adapters benefit from both the no-CPU-access and write-combine properties of default and upload heaps. See [GetCustomHeapProperties](#) for more details.

When **CacheCoherentUMA** is true, applications can more strongly entertain abandoning the attribution of heaps and using the custom heap equivalent of upload heaps everywhere. Zero-copy UMA optimizations such those offered by [WriteToSubresource](#) are more generally encouraged as more scenarios will just benefit from shared usage. The memory model is very conducive to more scenarios and wider adoption. Some corner cases may still exist where benefits are not easily obtained, but they should be much rarer and less detrimental than other options. For advanced developers: **CacheCoherentUMA** means that a significant amount of the caches in the memory hierarchy are also unified or integrated between the CPU and GPU. The most unique observable characteristic is that upload heaps are actually write-back on

CacheCoherentUMA. For these architecture, the usage of write-combine on upload heaps is commonly a detriment.

The low-level details should be ignored by a vast majority of single-adapter applications. As usual, single-adapter applications can simplify the landscape and ensure that the CPU writes to upload heaps use patterns that are write-combine-friendly. The lower-level details help reinforce the concepts for multi-adapter applications. Multi-adapter applications likely need to understand adapter architecture properties well enough to choose the optimal custom heap properties to efficiently move data between adapters.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY structure (d3d12.h)

Article 02/22/2024

Details the adapter's support for prioritization of different command queue types.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY {
    D3D12_COMMAND_LIST_TYPE CommandListType;
    UINT Priority;
    BOOL PriorityForTypeIsSupported;
} D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY;
```

Members

CommandListType

SAL: *In*

The type of the command list you're interested in.

Priority

SAL: *In*

The priority level you're interested in.

PriorityForTypeIsSupported

SAL: *out*

On return, contains true if the specified command list type supports the specified priority level; otherwise, false.

Remarks

Use this structure with [CheckFeatureSupport](#) to determine the priority levels supported by various command queue types.

See the enumeration constant `D3D12_FEATURE_COMMAND_QUEUE_PRIORITY` in the `D3D12_FEATURE` enumeration.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_CROSS_NODE structure (d3d12.h)

Article02/22/2024

Indicates the level of support for the sharing of resources between different adapters—for example, multiple GPUs.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_CROSS_NODE {
    D3D12 CROSS_NODE_SHARING_TIER SharingTier;
    BOOL AtomicShaderInstructions;
} D3D12_FEATURE_DATA_CROSS_NODE;
```

Members

SharingTier

Type: [D3D12_CROSS_NODE_SHARING_TIER](#)

Indicates the tier of cross-adapter sharing support.

AtomicShaderInstructions

Type: [BOOL](#)

Indicates there is support for shader instructions which operate across adapters.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS structure (d3d12.h)

Article04/02/2021

Describes Direct3D 12 feature options in the current graphics driver.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS {
    BOOL DoublePrecisionFloatShaderOps;
    BOOL OutputMergerLogicOp;
    D3D12_SHADER_MIN_PRECISION_SUPPORT MinPrecisionSupport;
    D3D12_TILED_RESOURCES_TIER TiledResourcesTier;
    D3D12_RESOURCE_BINDING_TIER ResourceBindingTier;
    BOOL PSSpecifiedStencilRefSupported;
    BOOL TypedUAVLoadAdditionalFormats;
    BOOL ROVsSupported;
    D3D12_CONSERVATIVE_RASTERIZATION_TIER ConservativeRasterizationTier;
    UINT MaxGPUVirtualAddressBitsPerResource;
    BOOL StandardSwizzle64KBSupported;
    D3D12_CROSS_NODE_SHARING_TIER CrossNodeSharingTier;
    BOOL CrossAdapterRowMajorTextureSupported;
    BOOL VPAndRTArrayIndexFromAnyShaderFeedingRasterizerSupportedWithoutGSEmulation;
    D3D12_RESOURCE_HEAP_TIER ResourceHeapTier;
} D3D12_FEATURE_DATA_D3D12_OPTIONS;
```

Members

DoublePrecisionFloatShaderOps

Specifies whether **double** types are allowed for shader operations. If **TRUE**, double types are allowed; otherwise **FALSE**. The supported operations are equivalent to Direct3D 11's **ExtendedDoublesShaderInstructions** member of the [D3D11_FEATURE_DATA_D3D11_OPTIONS](#) structure.

To use any HLSL shader that is compiled with a **double** type, the runtime must set **DoublePrecisionFloatShaderOps** to **TRUE**.

OutputMergerLogicOp

Specifies whether logic operations are available in blend state. The runtime sets this member to **TRUE** if logic operations are available in blend state and **FALSE** otherwise. This member is **FALSE** for feature level 9.1, 9.2, and 9.3. This member is optional for feature level 10, 10.1, and 11. This member is **TRUE** for feature level 11.1 and 12.

MinPrecisionSupport

A combination of [D3D12_SHADER_MIN_PRECISION_SUPPORT](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies minimum precision levels that the driver supports for shader stages. A value of zero indicates that the driver supports only full 32-bit precision for all shader stages.

TiledResourcesTier

Specifies whether the hardware and driver support tiled resources. The runtime sets this member to a [D3D12_TILED_RESOURCES_TIER](#)-typed value that indicates if the hardware and driver support tiled resources and at what tier level.

ResourceBindingTier

Specifies the level at which the hardware and driver support resource binding. The runtime sets this member to a [D3D12_RESOURCE_BINDING_TIER](#)-typed value that indicates the tier level.

PSSpecifiedStencilRefSupported

Specifies whether pixel shader stencil ref is supported. If **TRUE**, it's supported; otherwise **FALSE**.

TypedUAVLoadAdditionalFormats

Specifies whether the loading of additional formats for typed unordered-access views (UAVs) is supported. If **TRUE**, it's supported; otherwise **FALSE**.

ROVsSupported

Specifies whether [Rasterizer Order Views](#) (ROVs) are supported. If **TRUE**, they're supported; otherwise **FALSE**.

ConservativeRasterizationTier

Specifies the level at which the hardware and driver support conservative rasterization. The runtime sets this member to a [D3D12_CONSERVATIVE_RASTERIZATION_TIER](#)-typed value that indicates the tier level.

MaxGPUVirtualAddressBitsPerResource

Don't use this field; instead, use the [D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT](#) query (a structure with a **MaxGPUVirtualAddressBitsPerResource** member), which is more accurate.

StandardSwizzle64KBSupported

TRUE if the hardware supports textures with the 64KB standard swizzle pattern. Support for this pattern enables zero-copy texture optimizations while providing near-equilateral locality for each dimension within the texture. For texture swizzle options and restrictions, see [D3D12_TEXTURE_LAYOUT](#).

CrossNodeSharingTier

A [D3D12_CROSS_NODE_SHARING_TIER](#) enumeration constant that specifies the level of sharing across nodes of an adapter that has multiple nodes, such as Tier 1 Emulated, Tier 1, or Tier 2.

CrossAdapterRowMajorTextureSupported

FALSE means the device only supports copy operations to and from cross-adapter row-major textures. TRUE means the device supports shader resource views, unordered access views, and render target views of cross-adapter row-major textures. "Cross-adapter" means between multiple adapters (even from different IHVs).

VPAndRTArrayIndexFromAnyShaderFeedingRasterizerSupportedWithoutGSEmulation

Whether the viewport (VP) and Render Target (RT) array index from any shader feeding the rasterizer are supported without geometry shader emulation. Compare the **VPAndRTArrayIndexFromAnyShaderFeedingRasterizer** member of the [D3D11_FEATURE_DATA_D3D11_OPTIONS3](#) structure. In [ID3D12ShaderReflection::GetRequiresFlags](#), see the #define D3D_SHADER_REQUIRES_VIEWPORT_AND_RT_ARRAY_INDEX_FROM_ANY_SHADER_FEEDING_RASTERIZER.

ResourceHeapTier

Specifies the level at which the hardware and driver require heap attribution related to resource type. The runtime sets this member to a [D3D12_RESOURCE_HEAP_TIER](#) enumeration constant.

Remarks

See [D3D12_FEATURE](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Conservative Rasterization](#)

[Core Structures](#)

[D3D12_FEATURE](#)

[Rasterizer Ordered Views](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS

1 structure (d3d12.h)

Article 02/22/2024

Describes the level of support for HLSL 6.0 wave operations.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS1 {
    BOOL WaveOps;
    UINT WaveLaneCountMin;
    UINT WaveLaneCountMax;
    UINT TotalLaneCount;
    BOOL ExpandedComputeResourceStates;
    BOOL Int64ShaderOps;
} D3D12_FEATURE_DATA_D3D12_OPTIONS1;
```

Members

WaveOps

True if the driver supports HLSL 6.0 wave operations.

WaveLaneCountMin

Specifies the baseline number of lanes in the SIMD wave that this implementation can support. This term is sometimes known as "wavefront size" or "warp width". Currently apps should rely only on this minimum value for sizing workloads.

WaveLaneCountMax

Specifies the maximum number of lanes in the SIMD wave that this implementation can support.

TotalLaneCount

Specifies the total number of SIMD lanes on the hardware.

ExpandedComputeResourceStates

Indicates transitions are possible in and out of the CBV, and indirect argument states, on compute command lists. If [CheckFeatureSupport](#) succeeds this value will always be true.

Int64ShaderOps

Indicates that 64bit integer operations are supported.

Remarks

A "lane" is single thread of execution. The shader models before version 6.0 expose only one of these at the language level, leaving expansion to parallel SIMD processing entirely up to the implementation.

A "wave" is set of lanes (threads) executed simultaneously in the processor. No explicit barriers are required to guarantee that they execute in parallel. Similar concepts include "warp" and "wavefront".

This structure is used with the D3D12_FEATURE_D3D12_OPTIONS1 member of [D3D12_FEATURE](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS10 structure (d3d12.h)

Article02/22/2024

Indicates whether or not the SUM combiner can be used, and whether or not *SV_ShadingRate* can be set from a mesh shader.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS10 {
    BOOL VariableRateShadingSumCombinerSupported;
    BOOL MeshShaderPerPrimitiveShadingRateSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS10;
```

Members

`VariableRateShadingSumCombinerSupported`

Type: `_Out_ BOOL`

Indicates whether or not the SUM combiner can be used (this relates to [variable-rate shading](#) Tier 2). `true` if it can, otherwise `false`.

`MeshShaderPerPrimitiveShadingRateSupported`

Type: `_Out_ BOOL`

Indicates whether or not *SV_ShadingRate* can be set from a mesh shader (this relates to [variable-rate shading](#) Tier 2). `true` if it can, otherwise `false`.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows Build 22000

Requirement	Value
Minimum supported server	Windows Build 22000
Header	d3d12.h

See also

- [Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS11 structure (d3d12.h)

Article02/22/2024

Indicates whether or not 64-bit integer atomics on resources in descriptor heaps are supported.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS11 {
    BOOL AtomicInt64OnDescriptorHeapResourceSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS11;
```

Members

AtomicInt64OnDescriptorHeapResourceSupported

Type: `_Out_ BOOL`

Indicates whether or not 64-bit integer atomics on resources in descriptor heaps are supported. `true` if supported, otherwise `false`.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_FEATURE_DATA_D3D12_OPTIONS12 structure (d3d12.h)

Article02/22/2024

Indicates whether or not Enhanced Barriers are supported.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS12 {
    D3D12_TRI_STATE MSPrimitivesPipelineStatisticIncludesCulledPrimitives;
    BOOL EnhancedBarriersSupported;
    BOOL RelaxedFormatCastingSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS12;
```

Members

`MSPrimitivesPipelineStatisticIncludesCulledPrimitives`

Type: `_Out_ D3D12_TRI_STATE`

TBD

`EnhancedBarriersSupported`

Type: `_Out_ BOOL`

Indicates whether or not Enhanced Barriers are supported. `true` if supported, otherwise `false`.

Enhanced Barriers is not currently a hardware or driver requirement. So before using command list Barrier APIs, or resource creation APIs using the *InitialLayout* parameter, you must check for optional driver support via *EnhancedBarriersSupported*.

Requires the DirectX 12 Agility SDK 1.7 or later; otherwise, the value is always `FALSE`.

`RelaxedFormatCastingSupported`

Type: `_Out_ BOOL`

Technically used to indicate support for the functionality that enables integer aliasing.

Requires the DirectX 12 Agility SDK 1.7 or later; otherwise, the value is always FALSE.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 11, version 22H2; or DirectX 12 Agility SDK 1.6 or later
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS13 structure (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS13 {
    BOOL UnrestrictedBufferTextureCopyPitchSupported;
    BOOL UnrestrictedVertexElementAlignmentSupported;
    BOOL InvertedViewportHeightFlipsYSupported;
    BOOL InvertedViewportDepthFlipsZSupported;
    BOOL TextureCopyBetweenDimensionsSupported;
    BOOL AlphaBlendFactorSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS13;
```

Members

UnrestrictedBufferTextureCopyPitchSupported

Type: `_Out_ BOOL`

UnrestrictedVertexElementAlignmentSupported

Type: `_Out_ BOOL`

InvertedViewportHeightFlipsYSupported

Type: `_Out_ BOOL`

InvertedViewportDepthFlipsZSupported

Type: `_Out_ BOOL`

TextureCopyBetweenDimensionsSupported

Type: `_Out_ BOOL`

AlphaBlendFactorSupported

Type: `_Out_ BOOL`

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 11, version 22H2; or DirectX 12 Agility SDK 1.6 or later
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS2 structure (d3d12.h)

Article02/22/2024

Indicates the level of support that the adapter provides for depth-bounds tests and programmable sample positions.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS2 {
    BOOL DepthBoundsTestSupported;
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER ProgrammableSamplePositionsTier;
} D3D12_FEATURE_DATA_D3D12_OPTIONS2;
```

Members

DepthBoundsTestSupported

SAL: *Out*

On return, contains true if depth-bounds tests are supported; otherwise, false.

ProgrammableSamplePositionsTier

SAL: *Out*

On return, contains a value that indicates the level of support offered for programmable sample positions.

Remarks

Use this structure with [CheckFeatureSupport](#) to determine the level of support offered for the optional Depth-bounds test and programmable sample positions features.

See the enumeration constant D3D12_FEATURE_D3D12_OPTIONS2 in the [D3D12_FEATURE](#) enumeration.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS3 structure (d3d12.h)

Article04/02/2021

Indicates the level of support that the adapter provides for timestamp queries, format-casting, immediate write, view instancing, and barycentrics.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS3 {
    BOOL CopyQueueTimestampQueriesSupported;
    BOOL CastingFullyTypedFormatSupported;
    D3D12_COMMAND_LIST_SUPPORT_FLAGS WriteBufferImmediateSupportFlags;
    D3D12_VIEW_INSTANCING_TIER ViewInstancingTier;
    BOOL BarycentricsSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS3;
```

Members

`CopyQueueTimestampQueriesSupported`

Indicates whether timestamp queries are supported on copy queues.

`CastingFullyTypedFormatSupported`

Indicates whether casting from one fully typed format to another, compatible, format is supported.

`WriteBufferImmediateSupportFlags`

Indicates the kinds of command lists that support the ability to write an immediate value directly from the command stream into a specified buffer.

`ViewInstancingTier`

Indicates the level of support the adapter has for view instancing.

`BarycentricsSupported`

Indicates whether barycentrics are supported.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS4 structure (d3d12.h)

Article02/22/2024

Indicates the level of support for 64KB-aligned MSAA textures, cross-API sharing, and native 16-bit shader operations.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS4 {
    BOOL MSAA64KBAlignedTextureSupported;
    D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER SharedResourceCompatibilityTier;
    BOOL Native16BitShaderOpsSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS4;
```

Members

`MSAA64KBAlignedTextureSupported`

Type: [BOOL](#)

Indicates whether 64KB-aligned MSAA textures are supported.

`SharedResourceCompatibilityTier`

Type: [D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER](#)

Indicates the tier of cross-API sharing support.

`Native16BitShaderOpsSupported`

Type: [BOOL](#)

Indicates native 16-bit shader operations are supported. These operations require shader model 6_2. For more information, see the [16-Bit Scalar Types](#)  HLSL reference.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS5 structure (d3d12.h)

Article02/22/2024

Indicates the level of support that the adapter provides for render passes, ray tracing, and shader-resource view tier 3 tiled resources.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS5 {
    BOOL SRVOnlyTiledResourceTier3;
    D3D12_RENDER_PASS_TIER RenderPassesTier;
    D3D12_RAYTRACING_TIER RaytracingTier;
} D3D12_FEATURE_DATA_D3D12_OPTIONS5;
```

Members

`SRVOnlyTiledResourceTier3`

A boolean value indicating whether the options require shader-resource view tier 3 tiled resource support. For more information, see [D3D12_TILED_RESOURCES_TIER](#).

`RenderPassesTier`

The extent to which a device driver and/or the hardware efficiently supports render passes. See [D3D12_RENDERPASS_TIER](#).

RaytracingTier

Specifies the level of ray tracing support on the graphics device. See [D3D12_RAYTRACING_TIER](#).

`RaytracingTier`

Remarks

Pass [D3D12_FEATURE_D3D12_OPTIONS5](#) to [ID3D12Device::CheckFeatureSupport](#) to retrieve a [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS6 structure (d3d12.h)

Article 02/22/2024

Indicates the level of support that the adapter provides for variable-rate shading (VRS), and indicates whether or not background processing is supported. For more info, see [Variable-rate shading \(VRS\)](#), and the [Direct3D 12 background processing spec](#).

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS6 {
    BOOL AdditionalShadingRatesSupported;
    BOOL PerPrimitiveShadingRateSupportedWithViewportIndexing;
    D3D12_VARIABLE_SHADING_RATE_TIER VariableShadingRateTier;
    UINT ShadingRateImageTileSize;
    BOOL BackgroundProcessingSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS6;
```

Members

`AdditionalShadingRatesSupported`

Type: **BOOL**

Indicates whether 2x4, 4x2, and 4x4 coarse pixel sizes are supported for single-sampled rendering; and whether coarse pixel size 2x4 is supported for 2x MSAA. `true` if those sizes are supported, otherwise `false`.

`PerPrimitiveShadingRateSupportedWithViewportIndexing`

Type: **BOOL**

Indicates whether the per-provoking-vertex (also known as per-primitive) rate can be used with more than one viewport. If so, then, in that case, that rate can be used when `SV_VerIndex` is written to. `true` if that rate can be used with more than one viewport, otherwise `false`.

`VariableShadingRateTier`

Type: [D3D12_VARIABLE_SHADING_RATE_TIER](#)

Indicates the shading rate tier.

`ShadingRateImageTileSize`

Type: [UINT](#)

Indicates the tile size of the screen-space image as a [UINT](#).

`BackgroundProcessingSupported`

Type: [BOOL](#)

Indicates whether or not background processing is supported. `true` if background processing is supported, otherwise `false`. For more info, see the [Direct3D 12 background processing spec](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Variable-rate shading \(VRS\)](#)
- [Direct3D 12 background processing spec](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS7 structure (d3d12.h)

Article02/22/2024

Indicates the level of support that the adapter provides for mesh and amplification shaders, and for sampler feedback.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS7 {
    D3D12_MESH_SHADER_TIER      MeshShaderTier;
    D3D12_SAMPLER_FEEDBACK_TIER SamplerFeedbackTier;
} D3D12_FEATURE_DATA_D3D12_OPTIONS7;
```

Members

MeshShaderTier

Type: `_Out_ D3D12_MESH_SHADER_TIER`

Indicates the level of support for mesh and amplification shaders.

SamplerFeedbackTier

Type: `_Out_ D3D12_SAMPLER_FEEDBACK_TIER`

Indicates the level of support for sampler feedback.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Mesh shader spec ↗](#)
 - [Sampler feedback spec ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS8 structure (d3d12.h)

Article 10/20/2021

Indicates whether or not unaligned block-compressed textures are supported.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS8 {
    BOOL UnalignedBlockTexturesSupported;
} D3D12_FEATURE_DATA_D3D12_OPTIONS8;
```

Members

UnalignedBlockTexturesSupported

Type: `_Out_ BOOL`

Indicates whether or not unaligned block-compressed textures are supported.

If `false`, then Direct3D 12 requires that the dimensions of the top-level mip of a block-compressed texture are aligned to multiples of 4 (such alignment requirements do not apply to less-detailed mips). If `true`, then no such alignment requirement applies to any mip level of a block-compressed texture.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS9 structure (d3d12.h)

Article02/22/2024

Indicates whether or not support exists for mesh shaders, values of *SV_RenderTargetArrayIndex* that are 8 or greater, typed resource 64-bit integer atomics, derivative and derivative-dependent texture sample operations, and the level of support for WaveMMA (wave_matrix) operations.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_D3D12_OPTIONS9 {
    BOOL           MeshShaderPipelineStatsSupported;
    BOOL           MeshShaderSupportsFullRangeRenderTargetArrayIndex;
    BOOL           AtomicInt64OnTypedResourceSupported;
    BOOL           AtomicInt64OnGroupSharedSupported;
    BOOL           DerivativesInMeshAndAmplificationShadersSupported;
    D3D12_WAVE_MMA_TIER WaveMMATier;
} D3D12_FEATURE_DATA_D3D12_OPTIONS9;
```

Members

MeshShaderPipelineStatsSupported

Type: `_Out_ BOOL`

Indicates whether or not mesh shaders are supported. `true` if supported, otherwise `false`.

MeshShaderSupportsFullRangeRenderTargetArrayIndex

Type: `_Out_ BOOL`

Indicates whether or not values of *SV_RenderTargetArrayIndex* that are 8 or greater are supported. `true` if supported, otherwise `false`.

AtomicInt64OnTypedResourceSupported

Type: `_Out_ BOOL`

Indicates whether or not typed resource 64-bit integer atomics are supported. `true` if supported, otherwise `false`.

`AtomicInt64OnGroupSharedSupported`

Type: `_Out_ BOOL`

Indicates whether or not 64-bit integer atomics are supported on `groupshared` variables. `true` if supported, otherwise `false`.

`DerivativesInMeshAndAmplificationShadersSupported`

Type: `_Out_ BOOL`

Indicates whether or not derivative and derivative-dependent texture sample operations are supported. `true` if supported, otherwise `false`.

`WaveMMATier`

Type: `_Out_ D3D12_WAVE_MMA_TIER`

Indicates the level of support for WaveMMA (wave_matrix) operations.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Mesh shader spec ↗](#)

Feedback

Was this page helpful?

Yes

No

D3D12_FEATURE_DATA_DISPLAYABLE structure (d3d12.h)

Article02/22/2024

This feature is currently in preview.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_DISPLAYABLE {
    BOOL DisplayableTexture;
    D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER SharedResourceCompatibilityTier;
} D3D12_FEATURE_DATA_DISPLAYABLE;
```

Members

DisplayableTexture

SharedResourceCompatibilityTier

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Build 22000
Minimum supported server	Windows Build 22000
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_EXISTING_HEAP S structure (d3d12.h)

Article02/22/2024

Provides detail about whether the adapter supports creating heaps from existing system memory. Such heaps are not intended for general use, but are exceptionally useful for diagnostic purposes, because they are guaranteed to persist even after the adapter faults or experiences a device-removal event. Persistence is not guaranteed for heaps returned by [ID3D12Device::CreateHeap](#) or [ID3D12Device::CreateCommittedResource](#), even when the heap resides in system memory.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_EXISTING_HEAPS {
    BOOL Supported;
} D3D12_FEATURE_DATA_EXISTING_HEAPS;
```

Members

Supported

TRUE if the adapter can create a heap from existing system memory. Otherwise, FALSE.

Remarks

For a variety of performance and compatibility reasons, applications should not make use of this feature except for diagnostic purposes. In particular, heaps created using this feature only support system-memory heaps with cross-adapter properties, which precludes many optimization opportunities that typical application scenarios could otherwise take advantage of.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[ID3D12Device::CreateCommittedResource](#)

[ID3D12Device::CreateHeap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_FEATURE_LEVELS structure (d3d12.h)

Article02/22/2024

Describes info about the [feature levels](#) supported by the current graphics driver.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_FEATURE_LEVELS {
    UINT             NumFeatureLevels;
    const D3D_FEATURE_LEVEL *pFeatureLevelsRequested;
    D3D_FEATURE_LEVEL     MaxSupportedFeatureLevel;
} D3D12_FEATURE_DATA_FEATURE_LEVELS;
```

Members

NumFeatureLevels

The number of [feature levels](#) in the array at `pFeatureLevelsRequested`.

pFeatureLevelsRequested

A pointer to an array of [D3D_FEATURE_LEVELs](#) that the application is requesting for the driver and hardware to evaluate.

MaxSupportedFeatureLevel

The maximum [feature level](#) that the driver and hardware support.

Remarks

See [D3D12_FEATURE](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_FEATURE_DATA_FORMAT_INFO structure (d3d12.h)

Article 02/22/2024

Describes a DXGI data format and plane count.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_FORMAT_INFO {
    DXGI_FORMAT Format;
    UINT8 PlaneCount;
} D3D12_FEATURE_DATA_FORMAT_INFO;
```

Members

Format

A [DXGI_FORMAT](#)-typed value for the format to return info about.

PlaneCount

The number of planes to provide information about.

Remarks

See [D3D12_FEATURE](#).

Examples

C++

```
inline UINT8 D3D12GetFormatPlaneCount(
    _In_ ID3D12Device* pDevice,
    DXGI_FORMAT Format
)
{
    D3D12_FEATURE_DATA_FORMAT_INFO formatInfo{ Format };
    if (FAILED(pDevice->CheckFeatureSupport(D3D12_FEATURE_FORMAT_INFO,
        &formatInfo, sizeof(formatInfo))))
}
```

```
{  
    return 0;  
}  
return formatInfo.PlaneCount;  
}
```

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_FORMAT_SUPPORT structure (d3d12.h)

Article 02/22/2024

Describes which resources are supported by the current graphics driver for a given format.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_FORMAT_SUPPORT {
    DXGI_FORMAT Format;
    D3D12_FORMAT_SUPPORT1 Support1;
    D3D12_FORMAT_SUPPORT2 Support2;
} D3D12_FEATURE_DATA_FORMAT_SUPPORT;
```

Members

Format

A [DXGI_FORMAT](#)-typed value for the format to return info about.

Support1

A combination of [D3D12_FORMAT_SUPPORT1](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies which resources are supported.

Support2

A combination of [D3D12_FORMAT_SUPPORT2](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies which unordered resource options are supported.

Remarks

Refer to [Typed unordered access view loads](#) for an example use of this structure.

Also see [D3D12_FEATURE](#).

Hardware support for DXGI Formats

To view tables of DXGI formats and hardware features, refer to:

- [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](#)

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[Typed unordered access view loads](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT structure (d3d12.h)

Article04/02/2021

Details the adapter's GPU virtual address space limitations, including maximum address bits per resource and per process.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT {
    UINT MaxGPUVirtualAddressBitsPerResource;
    UINT MaxGPUVirtualAddressBitsPerProcess;
} D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT;
```

Members

MaxGPUVirtualAddressBitsPerResource

The maximum GPU virtual address bits per resource.

Some adapters have significantly less bits available per resource than per process, while other adapters have significantly greater bits available per resource than per process. The latter scenario tends to happen in less common scenarios, like when running a 32-bit process on certain UMA adapters. When per resource capabilities are greater than per process, the greater per resource capabilities can only be leveraged by reserved resources or NULL mapped pages.

MaxGPUVirtualAddressBitsPerProcess

The maximum GPU virtual address bits per process.

When this value is nearly equal to the available residency budget, [Evict](#) will not be a feasible option to manage residency. See [MakeResident](#) for more details.

Remarks

See the enumeration constant D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT in the [D3D12_FEATURE](#) enumeration.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS structure (d3d12.h)

Article 02/22/2024

Describes the multi-sampling image quality levels for a given format and sample count.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS {
    DXGI_FORMAT Format;
    UINT SampleCount;
    D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS Flags;
    UINT NumQualityLevels;
} D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS;
```

Members

Format

A [DXGI_FORMAT](#)-typed value for the format to return info about.

SampleCount

The number of multi-samples per pixel to return info about.

Flags

Flags to control quality levels, as a bitwise-OR'd combination of [D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS](#) enumeration constants. The resulting value specifies options for determining quality levels.

NumQualityLevels

The number of quality levels.

Remarks

See [D3D12_FEATURE](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT structure (d3d12.h)

Article02/22/2024

Indicates the level of support for protected resource sessions.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT {
    UINT NodeIndex;
    D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAGS Support;
} D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT;
```

Members

NodeIndex

Type: **UINT**

An input field, indicating the adapter index to query.

Support

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT structure (d3d12.h)

Article02/22/2024

Indicates a count of protected resource session types.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT {  
    UINT NodeIndex;  
    UINT Count;  
} D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPE_COUNT;
```

Members

NodeIndex

Type: [UINT](#)

An input parameter which, in multi-adapter operation, indicates which physical adapter of the device this operation applies to.

Count

Type: [UINT](#)

An output parameter containing the number of protected resource session types supported by the driver.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348

Requirement	Value
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES structure (d3d12.h)

Article02/22/2024

Indicates a list of protected resource session types.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES {  
    UINT NodeIndex;  
    UINT Count;  
    GUID *pTypes;  
} D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_TYPES;
```

Members

NodeIndex

Type: [UINT](#)

An input parameter which, in multi-adapter operation, indicates which physical adapter of the device this operation applies to.

Count

Type: [UINT](#)

An input parameter indicating the size of the *pTypes* array. This must match the count returned through the [D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPE_COUNT](#) query.

pTypes

Type: [GUID*](#)

An output parameter containing an array populated with the supported protected resource session types.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_QUERY_META_COMMAND structure (d3d12.h)

Article 02/22/2024

Indicates the level of support that the adapter provides for metacommands.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_QUERY_META_COMMAND {
    GUID          CommandId;
    UINT          NodeMask;
    const void*   pQueryInputData;
    SIZE_T        QueryInputDataSizeInBytes;
    void*         pQueryOutputData;
    SIZE_T        QueryOutputDataSizeInBytes;
} D3D12_FEATURE_DATA_QUERY_META_COMMAND;
```

Members

CommandId

Type: [GUID](#)

The fixed GUID that identifies the metacommand to query about.

NodeMask

Type: [UINT](#)

For single GPU operation, this is zero. If there are multiple GPU nodes, a bit is set to identify a node (the device's physical adapter). Each bit in the mask corresponds to a single node. Only 1 bit must be set. Refer to [Multi-adapter systems](#).

pQueryInputData

Type: [const void*](#)

A pointer to a buffer containing the query input data. Allocate *QueryInputDataSizeInBytes* bytes.

QueryInputDataSizeInBytes

Type: [SIZE_T](#)

The size of the buffer pointed to by *pQueryInputData*, in bytes.

`pQueryOutputData`

Type: [void*](#)

A pointer to a buffer containing the query output data.

`QueryOutputDataSizeInBytes`

Type: [SIZE_T](#)

The size of the buffer pointed to by *pQueryOutputData*, in bytes.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_ROOT_SIGNATURE structure (d3d12.h)

Article02/22/2024

Indicates root signature version support.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_ROOT_SIGNATURE {
    D3D_ROOT_SIGNATURE_VERSION HighestVersion;
} D3D12_FEATURE_DATA_ROOT_SIGNATURE;
```

Members

HighestVersion

On input, specifies the highest version [D3D_ROOT_SIGNATURE_VERSION](#) to check for.

On output specifies the highest version, up to the input version specified, actually available.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_SERIALIZATION structure (d3d12.h)

Article02/22/2024

Indicates the level of support for heap serialization.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_SERIALIZATION {
    UINT NodeIndex;
    D3D12_HEAP_SERIALIZATION_TIER HeapSerializationTier;
} D3D12_FEATURE_DATA_SERIALIZATION;
```

Members

NodeIndex

Type: [UINT](#)

An input field, indicating the adapter index to query.

HeapSerializationTier

Type: [D3D12_HEAP_SERIALIZATION_TIER](#)

An output field, indicating the tier of heap serialization support.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_SHADER_CACHE structure (d3d12.h)

Article02/22/2024

Describes the level of shader caching supported in the current graphics driver.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_SHADER_CACHE {
    D3D12_SHADER_CACHE_SUPPORT_FLAGS SupportFlags;
} D3D12_FEATURE_DATA_SHADER_CACHE;
```

Members

SupportFlags

Type: [D3D12_SHADER_CACHE_SUPPORT_FLAGS](#)

SAL: *out*

Indicates the level of caching supported.

Remarks

Use this structure with [CheckFeatureSupport](#) to determine the level of support offered for the optional shader-caching features.

See the enumeration constant D3D12_FEATURE_SHADER_CACHE in the [D3D12_FEATURE](#) enumeration.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FEATURE_DATA_SHADER_MODEL structure (d3d12.h)

Article02/22/2024

Contains the supported shader model.

Syntax

C++

```
typedef struct D3D12_FEATURE_DATA_SHADER_MODEL {
    D3D_SHADER_MODEL HighestShaderModel;
} D3D12_FEATURE_DATA_SHADER_MODEL;
```

Members

HighestShaderModel

Specifies one member of [D3D_SHADER_MODEL](#) that indicates the maximum supported shader model.

Remarks

Refer to the enumeration constant *D3D12_FEATURE_SHADER_MODEL* in the [D3D12_FEATURE](#).

When used with the [ID3D12Device::CheckFeatureSupport](#) function, before calling the function initialize the *HighestShaderModel* field to the highest shader model that your application understands. After the function completes successfully, the *HighestShaderModel* field contains the highest shader model that is both supported by the device and no higher than the shader model passed in.

ⓘ Note

[ID3D12Device::CheckFeatureSupport](#) returns [E_INVALIDARG](#) if *HighestShaderModel* isn't known by the current runtime. For that reason, we recommend that you call this in a loop with decreasing shader models to determine the highest supported shader model. Alternatively, use the caps

checking helper to simplify this; see the blog post [Introducing a New API for Checking Feature Support in Direct3D 12](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

- [Core structures](#)
- [D3D12_FEATURE](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FENCE_FLAGS enumeration (d3d12.h)

Article 07/27/2022

Specifies fence options.

Syntax

C++

```
typedef enum D3D12_FENCE_FLAGS {
    D3D12_FENCE_FLAG_NONE = 0,
    D3D12_FENCE_FLAG_SHARED = 0x1,
    D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER = 0x2,
    D3D12_FENCE_FLAG_NON_MONITORED = 0x4
};
```

Constants

[] Expand table

D3D12_FENCE_FLAG_NONE

Value: *0*

No options are specified.

D3D12_FENCE_FLAG_SHARED

Value: *0x1*

The fence is shared.

D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER

Value: *0x2*

The fence is shared with another GPU adapter.

D3D12_FENCE_FLAG_NON_MONITORED

Value: *0x4*

The fence is of the non-monitored type. Non-monitored fences should only be used when the adapter doesn't support monitored fences, or when a fence is shared with an adapter that doesn't support monitored fences.

Remarks

This enum is used by the [ID3D12Device::CreateFence](#) method.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_FILL_MODE enumeration (d3d12.h)

Article 02/22/2024

Specifies the fill mode to use when rendering triangles.

Syntax

C++

```
typedef enum D3D12_FILL_MODE {
    D3D12_FILL_MODE_WIREFRAME = 2,
    D3D12_FILL_MODE_SOLID = 3
};
```

Constants

 Expand table

D3D12_FILL_MODE_WIREFRAME

Value: 2

Draw lines connecting the vertices. Adjacent vertices are not drawn.

D3D12_FILL_MODE_SOLID

Value: 3

Fill the triangles formed by the vertices. Adjacent vertices are not drawn.

Remarks

Fill mode is specified in a [D3D12_RASTERIZER_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RASTERIZER_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_FILTER enumeration (d3d12.h)

Article01/31/2022

Specifies filtering options during texture sampling.

Syntax

C++

```
typedef enum D3D12_FILTER {
    D3D12_FILTER_MIN_MAG_MIP_POINT = 0,
    D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR = 0x1,
    D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x4,
    D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR = 0x5,
    D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT = 0x10,
    D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x11,
    D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT = 0x14,
    D3D12_FILTER_MIN_MAG_MIP_LINEAR = 0x15,
    D3D12_FILTER_MIN_MAG_ANISOTROPIC_MIP_POINT,
    D3D12_FILTER_ANISOTROPIC = 0x55,
    D3D12_FILTER_COMPARISON_MIN_MAG_MIP_POINT = 0x80,
    D3D12_FILTER_COMPARISON_MIN_MAG_POINT_MIP_LINEAR = 0x81,
    D3D12_FILTER_COMPARISON_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x84,
    D3D12_FILTER_COMPARISON_MIN_POINT_MAG_MIP_LINEAR = 0x85,
    D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_MIP_POINT = 0x90,
    D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x91,
    D3D12_FILTER_COMPARISON_MIN_MAG_LINEAR_MIP_POINT = 0x94,
    D3D12_FILTER_COMPARISON_MIN_MAG_MIP_LINEAR = 0x95,
    D3D12_FILTER_COMPARISON_MIN_MAG_ANISOTROPIC_MIP_POINT,
    D3D12_FILTER_COMPARISON_ANISOTROPIC = 0xd5,
    D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT = 0x100,
    D3D12_FILTER_MINIMUM_MIN_MAG_POINT_MIP_LINEAR = 0x101,
    D3D12_FILTER_MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x104,
    D3D12_FILTER_MINIMUM_MIN_POINT_MAG_MIP_LINEAR = 0x105,
    D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_MIP_POINT = 0x110,
    D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x111,
    D3D12_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT = 0x114,
    D3D12_FILTER_MINIMUM_MIN_MAG_MIP_LINEAR = 0x115,
    D3D12_FILTER_MINIMUM_MIN_MAG_ANISOTROPIC_MIP_POINT,
    D3D12_FILTER_MINIMUM_MIN_ANISOTROPIC = 0x155,
    D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT = 0x180,
    D3D12_FILTER_MAXIMUM_MIN_MAG_POINT_MIP_LINEAR = 0x181,
    D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT = 0x184,
    D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_MIP_LINEAR = 0x185,
    D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_MIP_POINT = 0x190,
    D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR = 0x191,
    D3D12_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT = 0x194,
    D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_LINEAR = 0x195,
    D3D12_FILTER_MAXIMUM_MIN_MAG_ANISOTROPIC_MIP_POINT,
```

```
D3D12_FILTER_MAXIMUM_ANISOTROPIC = 0x1d5  
} ;
```

Constants

[Expand table](#)

`D3D12_FILTER_MIN_MAG_MIP_POINT`

Value: *0*

Use point sampling for minification, magnification, and mip-level sampling.

`D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR`

Value: *0x1*

Use point sampling for minification and magnification; use linear interpolation for mip-level sampling.

`D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT`

Value: *0x4*

Use point sampling for minification; use linear interpolation for magnification; use point sampling for mip-level sampling.

`D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR`

Value: *0x5*

Use point sampling for minification; use linear interpolation for magnification and mip-level sampling.

`D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT`

Value: *0x10*

Use linear interpolation for minification; use point sampling for magnification and mip-level sampling.

`D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR`

Value: *0x11*

Use linear interpolation for minification; use point sampling for magnification; use linear interpolation for mip-level sampling.

`D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT`

Value: *0x14*

Use linear interpolation for minification and magnification; use point sampling for mip-level sampling.

`D3D12_FILTER_MIN_MAG_MIP_LINEAR`

Value: *0x15*

Use linear interpolation for minification, magnification, and mip-level sampling.

`D3D12_FILTER_ANISOTROPIC`

Value: *0x55*

Use anisotropic interpolation for minification, magnification, and mip-level sampling.

`D3D12_FILTER_COMPARISON_MIN_MAG_MIP_POINT`

Value: *0x80*

Use point sampling for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_MAG_POINT_MIP_LINEAR`

Value: *0x81*

Use point sampling for minification and magnification; use linear interpolation for mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_POINT_MAG_LINEAR_MIP_POINT`

Value: *0x84*

Use point sampling for minification; use linear interpolation for magnification; use point sampling for mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_POINT_MAG_MIP_LINEAR`

Value: *0x85*

Use point sampling for minification; use linear interpolation for magnification and mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_MIP_POINT`

Value: *0x90*

Use linear interpolation for minification; use point sampling for magnification and mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_POINT_MIP_LINEAR`

Value: *0x91*

Use linear interpolation for minification; use point sampling for magnification; use linear interpolation for mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_MAG_LINEAR_MIP_POINT`

Value: *0x94*

Use linear interpolation for minification and magnification; use point sampling for mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_MIN_MAG_MIP_LINEAR`

Value: *0x95*

Use linear interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

`D3D12_FILTER_COMPARISON_ANISOTROPIC`

Value: *0xd5*

Use anisotropic interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT

Value: 0x100

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_MIP_POINT](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_MAG_POINT_MIP_LINEAR

Value: 0x101

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT

Value: 0x104

Fetch the same set of texels as [D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_POINT_MAG_MIP_LINEAR

Value: 0x105

Fetch the same set of texels as [D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_MIP_POINT

Value: 0x110

Fetch the same set of texels as [D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR

Value: 0x111

Fetch the same set of texels as [D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT

Value: 0x114

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_MIN_MAG_MIP_LINEAR

Value: 0x715

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_MIP_LINEAR](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MINIMUM_ANISOTROPIC

Value: 0x155

Fetch the same set of texels as [D3D12_FILTER_ANISOTROPIC](#) and instead of filtering them return the minimum of the texels. Texels that are weighted 0 during filtering aren't counted towards the minimum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT

Value: 0x180

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_MIP_POINT](#) and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MAXIMUM_MIN_MAG_POINT_MIP_LINEAR

Value: 0x181

Fetch the same set of texels as [D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR](#) and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT

Value: 0x184

Fetch the same set of texels as [D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT](#) and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_MIP_LINEAR

Value: 0x185

Fetch the same set of texels as [D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR](#) and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_MIP_POINT

Value: 0x190

Fetch the same set of texels as [D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT](#) and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the **MinMaxFiltering** member in the [D3D11_FEATURE_DATA_D3D11_OPTIONS1](#) structure.

`D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR`

Value: `0x791`

Fetch the same set of texels as `D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR` and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the `MinMaxFiltering` member in the `D3D11_FEATURE_DATA_D3D11_OPTIONS1` structure.

`D3D12_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT`

Value: `0x194`

Fetch the same set of texels as `D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT` and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the `MinMaxFiltering` member in the `D3D11_FEATURE_DATA_D3D11_OPTIONS1` structure.

`D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_LINEAR`

Value: `0x195`

Fetch the same set of texels as `D3D12_FILTER_MIN_MAG_MIP_LINEAR` and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the `MinMaxFiltering` member in the `D3D11_FEATURE_DATA_D3D11_OPTIONS1` structure.

`D3D12_FILTER_MAXIMUM_ANISOTROPIC`

Value: `0x1d5`

Fetch the same set of texels as `D3D12_FILTER_ANISOTROPIC` and instead of filtering them return the maximum of the texels. Texels that are weighted 0 during filtering aren't counted towards the maximum. You can query support for this filter type from the `MinMaxFiltering` member in the `D3D11_FEATURE_DATA_D3D11_OPTIONS1` structure.

Remarks

This enum is used by the `D3D12_SAMPLER_DESC` structure.

Note If you use different filter types for min versus mag filter, undefined behavior occurs in certain cases where the choice between whether magnification or minification happens is ambiguous. To prevent this undefined behavior, use filter modes that use similar filter operations for both min and mag (or use anisotropic filtering, which avoids the issue as well).

During texture sampling, one or more texels are read and combined (this is called filtering) to produce a single value. Point sampling reads a single texel while linear sampling reads two texels (endpoints) and linearly interpolates a third value between the endpoints.

Microsoft High Level Shader Language (HLSL) texture-sampling functions also support comparison filtering during texture sampling. Comparison filtering compares each sampled texel against a comparison value. The boolean result is blended the same way that normal texture filtering is blended.

You can use HLSL intrinsic texture-sampling functions that implement texture filtering only or companion functions that use texture filtering with comparison filtering.

Also note the following defines:

syntax

```
#define D3D12_FILTER_REDUCTION_TYPE_MASK      ( 0x3 )

#define D3D12_FILTER_REDUCTION_TYPE_SHIFT     ( 7 )

#define D3D12_FILTER_TYPE_MASK    ( 0x3 )

#define D3D12_MIN_FILTER_SHIFT   ( 4 )

#define D3D12_MAG_FILTER_SHIFT   ( 2 )

#define D3D12_MIP_FILTER_SHIFT   ( 0 )

#define D3D12_ANISOTROPIC_FILTERING_BIT ( 0x40 )

#define D3D12_ENCODE_BASIC_FILTER( min, mag, mip, reduction )
 \
        ( ( D3D12_FILTER ) (
 \
            ( ( ( min ) & D3D12_FILTER_TYPE_MASK ) <<
D3D12_MIN_FILTER_SHIFT ) |
 \
            ( ( ( mag ) & D3D12_FILTER_TYPE_MASK ) <<
D3D12_MAG_FILTER_SHIFT ) |
 \
            ( ( ( mip ) & D3D12_FILTER_TYPE_MASK ) <<
D3D12_MIP_FILTER_SHIFT ) |
 \
            ( ( ( reduction ) &
D3D12_FILTER_REDUCTION_TYPE_MASK ) << D3D12_FILTER_REDUCTION_TYPE_SHIFT ) )
)

#define D3D12_ENCODE_ANISOTROPIC_FILTER( reduction )
 \
        ( ( D3D12_FILTER ) (
 \
            D3D12_ANISOTROPIC_FILTERING_BIT |
 \
            D3D12_ENCODE_BASIC_FILTER(
D3D12_FILTER_TYPE_LINEAR,           \
D3D12_FILTER_TYPE_LINEAR,           \
D3D12_FILTER_TYPE_LINEAR,           \

```

```

reduction ) ) )
#define D3D12_DECODE_MIN_FILTER( D3D12Filter )
\
( ( D3D12_FILTER_TYPE )
\
( ( ( D3D12Filter ) >>
D3D12_MIN_FILTER_SHIFT ) & D3D12_FILTER_TYPE_MASK ) )

#define D3D12_DECODE_MAG_FILTER( D3D12Filter )
\
( ( D3D12_FILTER_TYPE )
\
( ( ( D3D12Filter ) >>
D3D12_MAG_FILTER_SHIFT ) & D3D12_FILTER_TYPE_MASK ) )

#define D3D12_DECODE_MIP_FILTER( D3D12Filter )
\
( ( D3D12_FILTER_TYPE )
\
( ( ( D3D12Filter ) >>
D3D12_MIP_FILTER_SHIFT ) & D3D12_FILTER_TYPE_MASK ) )

#define D3D12_DECODE_FILTER_REDUCTION( D3D12Filter )
\
( ( D3D12_FILTER_REDUCTION_TYPE )
\
( ( ( D3D12Filter ) >>
D3D12_FILTER_REDUCTION_TYPE_SHIFT ) & D3D12_FILTER_REDUCTION_TYPE_MASK ) )

#define D3D12_DECODE_IS_COMPARISON_FILTER( D3D12Filter )
\
( D3D12_DECODE_FILTER_REDUCTION(
D3D12Filter ) == D3D12_FILTER_REDUCTION_TYPE_COMPARISON )

#define D3D12_DECODE_IS_ANISOTROPIC_FILTER( D3D12Filter )
\
( ( ( D3D12Filter ) &
D3D12_ANISOTROPIC_FILTERING_BIT ) && \
( D3D12_FILTER_TYPE_LINEAR ==
D3D12_DECODE_MIN_FILTER( D3D12Filter ) ) && \
( D3D12_FILTER_TYPE_LINEAR ==
D3D12_DECODE_MAG_FILTER( D3D12Filter ) ) && \
( D3D12_FILTER_TYPE_LINEAR ==
D3D12_DECODE_MIP_FILTER( D3D12Filter ) ) )

```

[] Expand table

Texture Sampling Function

Texture Sampling Function with Comparison Filtering

Sample

SampleCmp or SampleCmpLevelZero

Comparison filters only work with textures that have the following formats:

`DXGI_FORMAT_R32_FLOAT_X8X24_TYPELESS`, `DXGI_FORMAT_R32_FLOAT`,
`DXGI_FORMAT_R24_UNORM_X8_TYPELESS`, `DXGI_FORMAT_R16_UNORM`.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Descriptors](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FILTER_REDUCTION_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of filter reduction.

Syntax

C++

```
typedef enum D3D12_FILTER_REDUCTION_TYPE {
    D3D12_FILTER_REDUCTION_TYPE_STANDARD = 0,
    D3D12_FILTER_REDUCTION_TYPE_COMPARISON = 1,
    D3D12_FILTER_REDUCTION_TYPE_MINIMUM = 2,
    D3D12_FILTER_REDUCTION_TYPE_MAXIMUM = 3
};
```

Constants

[] Expand table

<code>D3D12_FILTER_REDUCTION_TYPE_STANDARD</code>

Value: 0

The filter type is standard.

<code>D3D12_FILTER_REDUCTION_TYPE_COMPARISON</code>

Value: 1

The filter type is comparison.

<code>D3D12_FILTER_REDUCTION_TYPE_MINIMUM</code>
--

Value: 2

The filter type is minimum.

<code>D3D12_FILTER_REDUCTION_TYPE_MAXIMUM</code>
--

Value: 3

The filter type is maximum.

Remarks

This enum is used by the [D3D12_SAMPLER_DESC](#) structure. Also, refer to the remarks for [D3D12_FILTER](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FILTER_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of magnification or minification sampler filters.

Syntax

C++

```
typedef enum D3D12_FILTER_TYPE {
    D3D12_FILTER_TYPE_POINT = 0,
    D3D12_FILTER_TYPE_LINEAR = 1
};
```

Constants

[] Expand table

<code>D3D12_FILTER_TYPE_POINT</code>

Value: 0

Point filtering is used as a texture magnification or minification filter. The texel with coordinates nearest to the desired pixel value is used. The texture filter to be used between mipmap levels is nearest-point mipmap filtering. The rasterizer uses the color from the texel of the nearest mipmap texture.

<code>D3D12_FILTER_TYPE_LINEAR</code>

Value: 1

Bilinear interpolation filtering is used as a texture magnification or minification filter. A weighted average of a 2 x 2 area of texels surrounding the desired pixel is used. The texture filter to use between mipmap levels is trilinear mipmap interpolation. The rasterizer linearly interpolates pixel color, using the texels of the two nearest mipmap textures.

Remarks

This enum is used by the [D3D12_SAMPLER_DESC](#) structure. Also, refer to the remarks for [D3D12_FILTER](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FORMAT_SUPPORT1 enumeration (d3d12.h)

Article01/31/2022

Specifies resources that are supported for a provided format.

Syntax

C++

```
typedef enum D3D12_FORMAT_SUPPORT1 {
    D3D12_FORMAT_SUPPORT1_NONE = 0,
    D3D12_FORMAT_SUPPORT1_BUFFER = 0x1,
    D3D12_FORMAT_SUPPORT1_IA_VERTEX_BUFFER = 0x2,
    D3D12_FORMAT_SUPPORT1_IA_INDEX_BUFFER = 0x4,
    D3D12_FORMAT_SUPPORT1_SO_BUFFER = 0x8,
    D3D12_FORMAT_SUPPORT1_TEXTURE1D = 0x10,
    D3D12_FORMAT_SUPPORT1_TEXTURE2D = 0x20,
    D3D12_FORMAT_SUPPORT1_TEXTURE3D = 0x40,
    D3D12_FORMAT_SUPPORT1_TEXTURECUBE = 0x80,
    D3D12_FORMAT_SUPPORT1_SHADER_LOAD = 0x100,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE = 0x200,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_COMPARISON = 0x400,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_MONO_TEXT = 0x800,
    D3D12_FORMAT_SUPPORT1_MIP = 0x1000,
    D3D12_FORMAT_SUPPORT1_RENDER_TARGET = 0x4000,
    D3D12_FORMAT_SUPPORT1_BLENDABLE = 0x8000,
    D3D12_FORMAT_SUPPORT1_DEPTH_STENCIL = 0x10000,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RESOLVE = 0x40000,
    D3D12_FORMAT_SUPPORT1_DISPLAY = 0x80000,
    D3D12_FORMAT_SUPPORT1_CAST_WITHIN_BIT_LAYOUT = 0x100000,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RENDERTARGET = 0x2000000,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_LOAD = 0x4000000,
    D3D12_FORMAT_SUPPORT1_SHADER_GATHER = 0x8000000,
    D3D12_FORMAT_SUPPORT1_BACK_BUFFER_CAST = 0x10000000,
    D3D12_FORMAT_SUPPORT1_TYPED_UNORDERED_ACCESS_VIEW = 0x20000000,
    D3D12_FORMAT_SUPPORT1_SHADER_GATHER_COMPARISON = 0x40000000,
    D3D12_FORMAT_SUPPORT1_DECODER_OUTPUT = 0x80000000,
    D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_OUTPUT = 0x100000000,
    D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_INPUT = 0x200000000,
    D3D12_FORMAT_SUPPORT1_VIDEO_ENCODER = 0x400000000
} ;
```

Constants

<code>D3D12_FORMAT_SUPPORT1_NONE</code> Value: <i>0</i> No resources are supported.
<code>D3D12_FORMAT_SUPPORT1_BUFFER</code> Value: <i>0x1</i> Buffer resources supported.
<code>D3D12_FORMAT_SUPPORT1_IA_VERTEX_BUFFER</code> Value: <i>0x2</i> Vertex buffers supported.
<code>D3D12_FORMAT_SUPPORT1_IA_INDEX_BUFFER</code> Value: <i>0x4</i> Index buffers supported.
<code>D3D12_FORMAT_SUPPORT1_SO_BUFFER</code> Value: <i>0x8</i> Streaming output buffers supported.
<code>D3D12_FORMAT_SUPPORT1_TEXTURE1D</code> Value: <i>0x10</i> 1D texture resources supported.
<code>D3D12_FORMAT_SUPPORT1_TEXTURE2D</code> Value: <i>0x20</i> 2D texture resources supported.
<code>D3D12_FORMAT_SUPPORT1_TEXTURE3D</code> Value: <i>0x40</i> 3D texture resources supported.
<code>D3D12_FORMAT_SUPPORT1_TEXTURECUBE</code> Value: <i>0x80</i> Cube texture resources supported.
<code>D3D12_FORMAT_SUPPORT1_SHADER_LOAD</code> Value: <i>0x100</i> The HLSL Load function for texture objects is supported.
<code>D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE</code> Value: <i>0x200</i> The HLSL Sample function for texture objects is supported.

Note If the device supports the format as a resource (1D, 2D, 3D, or cube map) but doesn't support this option, the resource can still use the [Sample](#) method but must use only the point filtering sampler state to perform the sample.

D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_COMPARISON

Value: *0x400*

The HLSL [SampleCmp](#) and [SampleCmpLevelZero](#) functions for texture objects are supported.

Note Windows 8 and later might provide limited support for these functions on Direct3D [feature levels](#) 9_1, 9_2, and 9_3. For more info, see [Implementing shadow buffers for Direct3D feature level 9](#).

D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_MONO_TEXT

Value: *0x800*

Reserved.

D3D12_FORMAT_SUPPORT1_MIP

Value: *0x1000*

Mipmaps are supported.

D3D12_FORMAT_SUPPORT1_RENDER_TARGET

Value: *0x4000*

Render targets are supported.

D3D12_FORMAT_SUPPORT1_BLENDABLE

Value: *0x8000*

Blend operations supported.

D3D12_FORMAT_SUPPORT1_DEPTH_STENCIL

Value: *0x10000*

Depth stencils supported.

D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RESOLVE

Value: *0x40000*

Multisample antialiasing (MSAA) resolve operations are supported. For more info, see [ID3D12GraphicsCommandList::ResolveSubresource](#).

D3D12_FORMAT_SUPPORT1_DISPLAY

Value: *0x80000*

Format can be displayed on screen.

D3D12_FORMAT_SUPPORT1_CAST_WITHIN_BIT_LAYOUT

Value: 0x1000000

Format can't be cast to another format.

D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RENDERTARGET

Value: 0x2000000

Format can be used as a multi-sampled render target.

D3D12_FORMAT_SUPPORT1_MULTISAMPLE_LOAD

Value: 0x4000000

Format can be used as a multi-sampled texture and read into a shader with the HLSL [Load](#) function.

D3D12_FORMAT_SUPPORT1_SHADER_GATHER

Value: 0x8000000

Format can be used with the HLSL gather function. This value is available in DirectX 10.1 or higher.

D3D12_FORMAT_SUPPORT1_BACK_BUFFER_CAST

Value: 0x1000000

Format supports casting when the resource is a back buffer.

D3D12_FORMAT_SUPPORT1_TYPED_UNORDERED_ACCESS_VIEW

Value: 0x2000000

Format can be used for an unordered access view.

D3D12_FORMAT_SUPPORT1_SHADER_GATHER_COMPARISON

Value: 0x4000000

Format can be used with the HLSL gather with comparison function.

D3D12_FORMAT_SUPPORT1_DECODER_OUTPUT

Value: 0x8000000

Format can be used with the decoder output.

D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_OUTPUT

Value: 0x10000000

Format can be used with the video processor output.

D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_INPUT

Value: 0x20000000

Format can be used with the video processor input.

D3D12_FORMAT_SUPPORT1_VIDEO_ENCODER

Value: 0x40000000

Format can be used with the video encoder.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_FORMAT_SUPPORT](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[D3D12_HEAP_FLAGS](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_FORMAT_SUPPORT2 enumeration (d3d12.h)

Article01/31/2022

Specifies which unordered resource options are supported for a provided format.

Syntax

C++

```
typedef enum D3D12_FORMAT_SUPPORT2 {
    D3D12_FORMAT_SUPPORT2_NONE = 0,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_ADD = 0x1,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS = 0x2,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE = 0x4,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE = 0x8,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX = 0x10,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX = 0x20,
    D3D12_FORMAT_SUPPORT2_UAV_TYPED_LOAD = 0x40,
    D3D12_FORMAT_SUPPORT2_UAV_TYPED_STORE = 0x80,
    D3D12_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP = 0x100,
    D3D12_FORMAT_SUPPORT2_TILED = 0x200,
    D3D12_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY = 0x4000,
    D3D12_FORMAT_SUPPORT2_SAMPLER_FEEDBACK
} ;
```

Constants

[] Expand table

`D3D12_FORMAT_SUPPORT2_NONE`

Value: *0*

No unordered resource options are supported.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_ADD`

Value: *0x1*

Format supports atomic add.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS`

Value: *0x2*

Format supports atomic bitwise operations.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE`

Value: `0x4`

Format supports atomic compare with store or exchange.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE`

Value: `0x8`

Format supports atomic exchange.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX`

Value: `0x10`

Format supports atomic min and max.

`D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX`

Value: `0x20`

Format supports atomic unsigned min and max.

`D3D12_FORMAT_SUPPORT2_UAV_TYPED_LOAD`

Value: `0x40`

Format supports a typed load.

`D3D12_FORMAT_SUPPORT2_UAV_TYPED_STORE`

Value: `0x80`

Format supports a typed store.

`D3D12_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP`

Value: `0x100`

Format supports logic operations in blend state.

`D3D12_FORMAT_SUPPORT2_TILED`

Value: `0x200`

Format supports tiled resources. Refer to [Volume Tiled Resources](#).

`D3D12_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY`

Value: `0x4000`

Format supports multi-plane overlays.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_FORMAT_SUPPORT](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GET_COARSE_SHADING_RATE_X_AXIS macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_GET_COARSE_SHADING_RATE_X_AXIS(  
    X  
);
```

Parameters

X

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗ | Get help at Microsoft Q&A

D3D12_GET_COARSE_SHADING_RATE_Y_AXIS macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_GET_COARSE_SHADING_RATE_Y_AXIS(  
    y  
);
```

Parameters

y

Return value

None

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GLOBAL_BARRIER structure (d3d12.h)

Article 02/22/2024

Describes a resource memory access barrier. Used by global, texture, and buffer barriers to indicate when resource memory must be made visible for a specific access type.

Syntax

C++

```
typedef struct D3D12_GLOBAL_BARRIER {
    D3D12_BARRIER_SYNC SyncBefore;
    D3D12_BARRIER_SYNC SyncAfter;
    D3D12_BARRIER_ACCESS AccessBefore;
    D3D12_BARRIER_ACCESS AccessAfter;
} D3D12_GLOBAL_BARRIER;
```

Members

SyncBefore

Synchronization scope of all preceding GPU work that must be completed before executing the barrier.

SyncAfter

Synchronization scope of all subsequent GPU work that must wait until the barrier execution is finished.

AccessBefore

Access bits corresponding with any relevant resource usage since the preceding barrier or the start of **ExecuteCommandLists** scope.

AccessAfter

Access bits corresponding with any relevant resource usage after the barrier completes.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_GLOBAL_ROOT_SIGNATURE structure (d3d12.h)

Article04/02/2021

Defines a global root signature state subobject that will be used with associated shaders.

Syntax

C++

```
typedef struct D3D12_GLOBAL_ROOT_SIGNATURE {
    ID3D12RootSignature *pGlobalRootSignature;
} D3D12_GLOBAL_ROOT_SIGNATURE;
```

Members

`pGlobalRootSignature`

The root signature that will function as a global root signature. A state object holds a reference to this signature.

Remarks

The presence of this subobject in a state object is optional. The combination of global and/or local root signatures associated with any given shader function must define all resource bindings declared by the shader with no overlap across global and local root signatures.

If any given function in a call graph is associated with a particular global root signature, any other functions in the graph must either be associated with the same global root signature or none, and the shader entry (the root of the call graph) must be associated with the global root signature.

Different shaders can use different global root signatures (or none) within a state object, however any shaders referenced during a particular [DispatchRays](#) operation from a command list must have specified the same global root signature as what has been set on the command list as the compute root signature. So it is valid to define a single large state object with multiple global root signatures associated with different subsets of the

shaders. Apps are not forced to split their state object just because some shaders use different global root signatures.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GPU_DESCRIPTOR_HANDLE structure (d3d12.h)

Article02/22/2024

Describes a GPU descriptor handle.

Syntax

C++

```
typedef struct D3D12_GPU_DESCRIPTOR_HANDLE {
    UINT64 ptr;
} D3D12_GPU_DESCRIPTOR_HANDLE;
```

Members

ptr

The address of the descriptor.

Remarks

This structure is returned by

[ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart](#).

This structure is passed into the following methods:

- [ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat](#)
- [ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint](#)
- [ID3D12GraphicsCommandList::SetComputeRootDescriptorTable](#)
- [ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable](#)

To get the handle increment size use [ID3D12Device.GetDescriptorHandleIncrementSize](#)

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_GPU_DESCRIPTOR_HANDLE ID3D12Device.GetDescriptorHandleIncrementSize](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE structure (d3d12.h)

Article 02/22/2024

Represents a GPU virtual address and indexing stride.

Syntax

C++

```
typedef struct D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE {
    D3D12_GPU_VIRTUAL_ADDRESS StartAddress;
    UINT64                  StrideInBytes;
} D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE;
```

Members

`StartAddress`

The beginning of the virtual address range.

`StrideInBytes`

Defines indexing stride, such as for vertices. Only the bottom 32 bits are used. The field is 64 bits to make alignment of containing structures consistent everywhere.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

D3D12_GPU_VIRTUAL_ADDRESS_RANGE structure (d3d12.h)

Article02/22/2024

Represents a GPU virtual address range.

Syntax

C++

```
typedef struct D3D12_GPU_VIRTUAL_ADDRESS_RANGE {
    D3D12_GPU_VIRTUAL_ADDRESS StartAddress;
    UINT64                  SizeInBytes;
} D3D12_GPU_VIRTUAL_ADDRESS_RANGE;
```

Members

`StartAddress`

The beginning of the virtual address range.

`SizeInBytes`

The size of the virtual address range, in bytes.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE structure (d3d12.h)

Article 02/22/2024

Represents a GPU virtual address range and stride.

Syntax

C++

```
typedef struct D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE {
    D3D12_GPU_VIRTUAL_ADDRESS StartAddress;
    UINT64                  SizeInBytes;
    UINT64                  StrideInBytes;
} D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE;
```

Members

StartAddress

The beginning of the virtual address range.

SizeInBytes

The size of the virtual address range, in bytes.

StrideInBytes

Defines the record-indexing stride within the memory range.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GRAPHICS_PIPELINE_STATE_DESC

C structure (d3d12.h)

Article 09/01/2022

Describes a graphics pipeline state object.

Syntax

C++

```
typedef struct D3D12_GRAPHICS_PIPELINE_STATE_DESC {
    ID3D12RootSignature             *pRootSignature;
    D3D12_SHADER_BYTECODE          VS;
    D3D12_SHADER_BYTECODE          PS;
    D3D12_SHADER_BYTECODE          DS;
    D3D12_SHADER_BYTECODE          HS;
    D3D12_SHADER_BYTECODE          GS;
    D3D12_STREAM_OUTPUT_DESC       StreamOutput;
    D3D12_BLEND_DESC                BlendState;
    UINT                           SampleMask;
    D3D12_RASTERIZER_DESC          RasterizerState;
    D3D12_DEPTH_STENCIL_DESC       DepthStencilState;
    D3D12_INPUT_LAYOUT_DESC         InputLayout;
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE IBStripCutValue;
    D3D12_PRIMITIVE_TOPOLOGY_TYPE   PrimitiveTopologyType;
    UINT                           NumRenderTargets;
    DXGI_FORMAT                    RTVFormats[8];
    DXGI_FORMAT                    DSVFormat;
    DXGI_SAMPLE_DESC                SampleDesc;
    UINT                           NodeMask;
    D3D12_CACHED_PIPELINE_STATE    CachedPSO;
    D3D12_PIPELINE_STATE_FLAGS      Flags;
} D3D12_GRAPHICS_PIPELINE_STATE_DESC;
```

Members

pRootSignature

A pointer to the [ID3D12RootSignature](#) object.

VS

A [D3D12_SHADER_BYTECODE](#) structure that describes the vertex shader.

PS

A [D3D12_SHADER_BYTECODE](#) structure that describes the pixel shader.

DS

A [D3D12_SHADER_BYTECODE](#) structure that describes the domain shader.

HS

A [D3D12_SHADER_BYTECODE](#) structure that describes the hull shader.

GS

A [D3D12_SHADER_BYTECODE](#) structure that describes the geometry shader.

StreamOutput

A [D3D12_STREAM_OUTPUT_DESC](#) structure that describes a streaming output buffer.

BlendState

A [D3D12_BLEND_DESC](#) structure that describes the blend state.

SampleMask

The sample mask for the blend state.

RasterizerState

A [D3D12_RASTERIZER_DESC](#) structure that describes the rasterizer state.

DepthStencilState

A [D3D12_DEPTH_STENCIL_DESC](#) structure that describes the depth-stencil state.

InputLayout

A [D3D12_INPUT_LAYOUT_DESC](#) structure that describes the input-buffer data for the input-assembler stage.

IBStripCutValue

Specifies the properties of the index buffer in a [D3D12_INDEX_BUFFER_STRIP_CUT_VALUE](#) structure.

PrimitiveTopologyType

A [D3D12_PRIMITIVE_TOPOLOGY_TYPE](#)-typed value for the type of primitive, and ordering of the primitive data.

NumRenderTargets

The number of render target formats in the **RTVFormats** member.

RTVFormats[8]

An array of [DXGI_FORMAT](#)-typed values for the render target formats.

DSVFormat

A [DXGI_FORMAT](#)-typed value for the depth-stencil format.

SampleDesc

A [DXGI_SAMPLE_DESC](#) structure that specifies multisampling parameters.

NodeMask

For single GPU operation, set this to zero. If there are multiple GPU nodes, set bits to identify the nodes (the device's physical adapters) for which the graphics pipeline state is to apply. Each bit in the mask corresponds to a single node. Refer to [Multi-adapter systems](#).

CachedPSO

A cached pipeline state object, as a [D3D12_CACHED_PIPELINE_STATE](#) structure.
pCachedBlob and CachedBlobSizeInBytes may be set to NULL and 0 respectively.

Flags

A [D3D12_PIPELINE_STATE_FLAGS](#) enumeration constant such as for "tool debug".

Remarks

This structure is used by the [CreateGraphicsPipelineState](#) method.

The runtime validates:

- Whether the linkage between the shader stages is correct.
- If the **HS** and **DS** members are specified, the **PrimitiveTopologyType** member for topology type must be set to [D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH](#).
- Whether sample frequency execution isn't allowed with the center multi-sample anti-aliasing (MSAA) pattern.
- Whether anti-aliasing lines aren't allowed with the center MSAA pattern.

- If the **ForcedSampleCount** member of [D3D12_RASTERIZER_DESC](#) that **RasterizerState** specifies isn't zero:
 - Depth/stencil must be disabled.
 - Pixel shader can't output depth.
 - Pixel shader can't run at sample frequency.
 - Render target sample count must be 1.
- Whether blend state is compatible with render target formats.
- Whether pixel shader output type is compatible with render target format.
- Whether the sample count and quality are supported for the render target/depth stencil formats.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Conservative Rasterization](#)

[Core Structures](#)

[Rasterizer Ordered Views](#)

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GRAPHICS_STATES enumeration (d3d12.h)

Article01/31/2022

Defines flags that specify states related to a graphics command list. Values can be bitwise OR'd together.

Syntax

C++

```
typedef enum D3D12_GRAPHICS_STATES {
    D3D12_GRAPHICS_STATE_NONE = 0,
    D3D12_GRAPHICS_STATE_IA_VERTEX_BUFFERS,
    D3D12_GRAPHICS_STATE_IA_INDEX_BUFFER,
    D3D12_GRAPHICS_STATE_IA_PRIMITIVE_TOPOLOGY,
    D3D12_GRAPHICS_STATE_DESCRIPTOR_HEAP,
    D3D12_GRAPHICS_STATE_GRAPHICS_ROOT_SIGNATURE,
    D3D12_GRAPHICS_STATE_COMPUTE_ROOT_SIGNATURE,
    D3D12_GRAPHICS_STATE_RS_VIEWPORTS,
    D3D12_GRAPHICS_STATE_RS_SCISSOR_RECTS,
    D3D12_GRAPHICS_STATE_PREDICATION,
    D3D12_GRAPHICS_STATE_OM_RENDER_TARGETS,
    D3D12_GRAPHICS_STATE_OM_STENCIL_REF,
    D3D12_GRAPHICS_STATE_OM_BLEND_FACTOR,
    D3D12_GRAPHICS_STATE_PIPELINE_STATE,
    D3D12_GRAPHICS_STATE_SO_TARGETS,
    D3D12_GRAPHICS_STATE_OM_DEPTH_BOUNDS,
    D3D12_GRAPHICS_STATE_SAMPLE_POSITIONS,
    D3D12_GRAPHICS_STATE_VIEW_INSTANCE_MASK
} ;
```

Constants

[] Expand table

<code>D3D12_GRAPHICS_STATE_NONE</code>
Value: 0
Specifies no state.

<code>D3D12_GRAPHICS_STATE_IA_VERTEX_BUFFERS</code>
Specifies the state of the vertex buffer bindings on the input assembler stage.

D3D12_GRAPHICS_STATE_IA_INDEX_BUFFER
Specifies the state of the index buffer binding on the input assembler stage.
D3D12_GRAPHICS_STATE_IA_PRIMITIVE_TOPOLOGY
Specifies the state of the primitive topology value set on the input assembler stage.
D3D12_GRAPHICS_STATE_DESCRIPTOR_HEAP
Specifies the state of the currently bound descriptor heaps.
D3D12_GRAPHICS_STATE_GRAPHICS_ROOT_SIGNATURE
Specifies the state of the currently set graphics root signature.
D3D12_GRAPHICS_STATE_COMPUTE_ROOT_SIGNATURE
Specifies the state of the currently set compute root signature.
D3D12_GRAPHICS_STATE_RS_VIEWPORTS
Specifies the state of the viewports bound to the rasterizer stage.
D3D12_GRAPHICS_STATE_RS_SCISSOR_RECTS
Specifies the state of the scissor rectangles bound to the rasterizer stage.
D3D12_GRAPHICS_STATE_PREDICATION
Specifies the predicate state.
D3D12_GRAPHICS_STATE_OM_RENDER_TARGETS
Specifies the state of the render targets bound to the output merger stage.
D3D12_GRAPHICS_STATE_OM_STENCIL_REF
Specifies the state of the reference value for depth stencil tests set on the output merger stage.
D3D12_GRAPHICS_STATE_OM_BLEND_FACTOR
Specifies the state of the blend factor set on the output merger stage.
D3D12_GRAPHICS_STATE_PIPELINE_STATE
Specifies the state of the pipeline state object.
D3D12_GRAPHICS_STATE_SO_TARGETS
Specifies the state of the buffer views bound to the stream output stage.
D3D12_GRAPHICS_STATE_OM_DEPTH_BOUNDS
Specifies the state of the depth bounds set on the output merger stage.
D3D12_GRAPHICS_STATE_SAMPLE_POSITIONS
Specifies the state of the sample positions.
D3D12_GRAPHICS_STATE_VIEW_INSTANCE_MASK
Specifies the state of the view instances mask.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_HEAP_DESC structure (d3d12.h)

Article 04/02/2021

Describes a heap.

Syntax

```
C++  
  
typedef struct D3D12_HEAP_DESC {  
    UINT64           SizeInBytes;  
    D3D12_HEAP_PROPERTIES Properties;  
    UINT64           Alignment;  
    D3D12_HEAP_FLAGS   Flags;  
} D3D12_HEAP_DESC;
```

Members

SizeInBytes

The size, in bytes, of the heap. To avoid wasting memory, applications should pass *SizeInBytes* values which are multiples of the effective *Alignment*; but non-aligned *SizeInBytes* is also supported, for convenience. To find out how large a heap must be to support textures with undefined layouts and adapter-specific sizes, call [ID3D12Device::GetResourceAllocationInfo](#).

Properties

A [D3D12_HEAP_PROPERTIES](#) structure that describes the heap properties.

Alignment

The alignment value for the heap. Valid values:

Expand table

Value	Description
0	An alias for 64KB.
D3D12_DEFAULT_RESOURCE_PLACEMENT_ALIGNMENT	#defined as 64KB.
D3D12_DEFAULT_MSAA_RESOURCE_PLACEMENT_ALIGNMENT	#defined as 4MB. An application must decide whether the heap will contain multi-sample anti-aliasing (MSAA), in which case, the application must choose D3D12_DEFAULT_MSAA_RESOURCE_PLACEMENT_ALIGNMENT.

Flags

A combination of [D3D12_HEAP_FLAGS](#)-typed values that are combined by using a bitwise-OR operation. The resulting value identifies heap options. When creating heaps to support adapters with resource heap tier 1, an application must choose some flags.

Remarks

This structure is used by the [CreateHeap](#) method, and returned by the [GetDesc](#) method.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_HEAP_DESC](#)

[Core Structures](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HEAP_FLAGS enumeration (d3d12.h)

Article 02/14/2023

Specifies heap options, such as whether the heap can contain textures, and whether resources are shared across adapters.

Syntax

C++

```
typedef enum D3D12_HEAP_FLAGS {
    D3D12_HEAP_FLAG_NONE = 0,
    D3D12_HEAP_FLAG_SHARED = 0x1,
    D3D12_HEAP_FLAG_DENY_BUFFERS = 0x4,
    D3D12_HEAP_FLAG_ALLOW_DISPLAY = 0x8,
    D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER = 0x20,
    D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES = 0x40,
    D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES = 0x80,
    D3D12_HEAP_FLAG_HARDWARE_PROTECTED = 0x100,
    D3D12_HEAP_FLAG_ALLOW_WRITE_WATCH = 0x200,
    D3D12_HEAP_FLAG_ALLOW_SHADER_ATOMICS = 0x400,
    D3D12_HEAP_FLAG_CREATE_NOT_RESIDENT = 0x800,
    D3D12_HEAP_FLAG_CREATE_NOT_ZEROED = 0x1000,
    D3D12_HEAP_FLAG_TOOLS_USE_MANUAL_WRITE_TRACKING,
    D3D12_HEAP_FLAG_ALLOW_ALL_BUFFERS_AND_TEXTURES = 0,
    D3D12_HEAP_FLAG_ALLOW_ONLY_BUFFERS = 0xc0,
    D3D12_HEAP_FLAG_ALLOW_ONLY_NON_RT_DS_TEXTURES = 0x44,
    D3D12_HEAP_FLAG_ALLOW_ONLY_RT_DS_TEXTURES = 0x84
} ;
```

Constants

[] Expand table

`D3D12_HEAP_FLAG_NONE`

Value: *0*

No options are specified.

`D3D12_HEAP_FLAG_SHARED`

Value: *0x1*

The heap is shared. Refer to [Shared Heaps](#).

D3D12_HEAP_FLAG_DENY_BUFFERS

Value: 0x4

The heap isn't allowed to contain buffers.

D3D12_HEAP_FLAG_ALLOW_DISPLAY

Value: 0x8

The heap is allowed to contain swap-chain surfaces.

D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER

Value: 0x20

The heap is allowed to share resources across adapters. Refer to [Shared Heaps](#). A protected session cannot be mixed with resources that are shared across adapters.

D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES

Value: 0x40

The heap is not allowed to store Render Target (RT) and/or Depth-Stencil (DS) textures.

D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES

Value: 0x80

The heap is not allowed to contain resources with D3D12_RESOURCE_DIMENSION_TEXTURE1D, D3D12_RESOURCE_DIMENSION_TEXTURE2D, or D3D12_RESOURCE_DIMENSION_TEXTURE3D unless either D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET or D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL are present. Refer to [D3D12_RESOURCE_DIMENSION](#) and [D3D12_RESOURCE_FLAGS](#).

D3D12_HEAP_FLAG_HARDWARE_PROTECTED

Value: 0x100

Unsupported. Do not use.

D3D12_HEAP_FLAG_ALLOW_WRITE_WATCH

Value: 0x200

The heap supports MEM_WRITE_WATCH functionality, which causes the system to track the pages that are written to in the committed memory region. This flag can't be combined with the D3D12_HEAP_TYPE_DEFAULT or D3D12_CPU_PAGE_PROPERTY_UNKNOWN flags. Applications are discouraged from using this flag themselves because it prevents tools from using this functionality.

D3D12_HEAP_FLAG_ALLOW_SHADER_ATOMICS

Value: 0x400

Ensures that atomic operations will be atomic on this heap's memory, according to components able to see the memory.

Creating a heap with this flag will fail under either of these conditions.

- The heap type is D3D12_HEAP_TYPE_DEFAULT, and the heap can be visible on multiple nodes, but the device does *not* support [D3D12_CROSS_NODE_SHARING_TIER_3](#).
- The heap is CPU-visible, but the heap type is *not* D3D12_HEAP_TYPE_CUSTOM.

Note that heaps with this flag might be a limited resource on some systems.

`D3D12_HEAP_FLAG_CREATE_NOT_RESIDENT`

Value: *0x800*

The heap is created in a non-resident state and must be made resident using [ID3D12Device::MakeResident](#) or [ID3D12Device3::EnqueueMakeResident](#).

By default, the final step of heap creation is to make the heap resident, so this flag skips this step and allows the application to decide when to do so.

`D3D12_HEAP_FLAG_CREATE_NOT_ZEROED`

Value: *0x1000*

Allows the OS to not zero the heap created. By default, committed resources and heaps are almost always zeroed upon creation. This flag allows this to be elided in some scenarios. However, it doesn't guarantee it. For example, memory coming from other processes still needs to be zeroed for data protection and process isolation. This can lower the overhead of creating the heap.

`D3D12_HEAP_FLAG_ALLOW_ALL_BUFFERS_AND_TEXTURES`

Value: *0*

The heap is allowed to store all types of buffers and/or textures. This is an alias; for more details, see "Aliases" in the Remarks section.

`D3D12_HEAP_FLAG_ALLOW_ONLY_BUFFERS`

Value: *0xc0*

The heap is only allowed to store buffers. This is an alias; for more details, see "Aliases" in the Remarks section.

`D3D12_HEAP_FLAG_ALLOW_ONLY_NON_RT_DS_TEXTURES`

Value: *0x44*

The heap is only allowed to store non-RT, non-DS textures. This is an alias; for more details, see "Aliases" in the Remarks section.

`D3D12_HEAP_FLAG_ALLOW_ONLY_RT_DS_TEXTURES`

Value: *0x84*

The heap is only allowed to store RT and/or DS textures. This is an alias; for more details, see "Aliases" in the Remarks section.

Remarks

This enum is used by the following API items:

- [ID3D12Device::CreateHeap](#)
- [ID3D12Device::CreateCommittedResource](#)
- [D3D12_HEAP_DESC](#) structure

The following heap flags must be used with [ID3D12Device::CreateHeap](#), but will be set automatically for implicit heaps created by [ID3D12Device::CreateCommittedResource](#).

Adapters that only support [heap tier 1](#) must set two out of the three following flags.

[Expand table](#)

Value	Description
D3D12_HEAP_FLAG_DENY_BUFFERS	The heap isn't allowed to contain resources with D3D12_RESOURCE_DIMENSION_BUFFER (which is a D3D12_RESOURCE_DIMENSION enumeration constant).
D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES	The heap isn't allowed to contain resources with D3D12_RESOURCE_DIMENSION_TEXTURE1D, D3D12_RESOURCE_DIMENSION_TEXTURE2D, or D3D12_RESOURCE_DIMENSION_TEXTURE3D together with either D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET or D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL. (The latter two items are D3D12_RESOURCE_FLAGS enumeration constants.)
D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES	The heap isn't allowed to contain resources with D3D12_RESOURCE_DIMENSION_TEXTURE1D, D3D12_RESOURCE_DIMENSION_TEXTURE2D, or D3D12_RESOURCE_DIMENSION_TEXTURE3D unless D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET and D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL are absent.

Aliases

Adapters that support [heap tier 2](#) or greater are additionally allowed to set none of the above flags. Aliases for these flags are available for applications that prefer thinking only of which resources are supported.

The following aliases exist, so be careful when doing bit-manipulations:

- D3D12_HEAP_FLAG_ALLOW_ALL_BUFFERS_AND_TEXTURES = 0 and is only supported on [heap tier 2](#) and greater.
- D3D12_HEAP_FLAG_ALLOW_ONLY_BUFFERS = D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES | D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES

- D3D12_HEAP_FLAG_ALLOW_ONLY_NON_RT_DS_TEXTURES =
D3D12_HEAP_FLAG_DENY_BUFFERS | D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES
- D3D12_HEAP_FLAG_ALLOW_ONLY_RT_DS_TEXTURES =
D3D12_HEAP_FLAG_DENY_BUFFERS |
D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES

Displayable heaps

Displayable heaps are most commonly created by the swapchain for presentation, to enable scanning out to a monitor.

Displayable heaps are specified with the D3D12_HEAP_FLAG_ALLOW_DISPLAY member of the [D3D12_HEAP_FLAGS](#) enum.

Applications may create displayable heaps outside of a swapchain; but cannot actually present with them. This flag is not supported by [CreateHeap](#) and can only be used with [CreateCommittedResource](#) with D3D12_HEAP_TYPE_DEFAULT.

Additional restrictions to the [D3D12_RESOURCE_DESC](#) apply to the resource created with displayable heaps.

- The format must not only be supported by the device, but must be supported for scan-out. Refer to the use of the D3D12_FORMAT_SUPPORT1_DISPLAY member of [D3D12_FORMAT_SUPPORT1](#).
- *Dimension* must be D3D12_RESOURCE_DIMENSION_TEXTURE2D.
- *Alignment* must be 0.
- *ArraySize* may be either 1 or 2.
- *MipLevels* must be 1.
- *SampleDesc* must have *Count* set to 1 and *Quality* set to 0.
- *Layout* must be D3D12_TEXTURE_LAYOUT_UNKNOWN.
- D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL and D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER are invalid flags.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_HEAP_DESC](#)

[Core Enumerations](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_HEAP_PROPERTIES structure (d3d12.h)

Article04/02/2021

Describes heap properties.

Syntax

C++

```
typedef struct D3D12_HEAP_PROPERTIES {
    D3D12_HEAP_TYPE          Type;
    D3D12_CPU_PAGE_PROPERTY CPUPageProperty;
    D3D12_MEMORY_POOL        MemoryPoolPreference;
    UINT                      CreationNodeMask;
    UINT                      VisibleNodeMask;
} D3D12_HEAP_PROPERTIES;
```

Members

Type

A [D3D12_HEAP_TYPE](#)-typed value that specifies the type of heap.

CPUPageProperty

A [D3D12_CPU_PAGE_PROPERTY](#)-typed value that specifies the CPU-page properties for the heap.

MemoryPoolPreference

A [D3D12_MEMORY_POOL](#)-typed value that specifies the memory pool for the heap.

CreationNodeMask

For multi-adapter operation, this indicates the node where the resource should be created.

Exactly one bit of this **UINT** must be set. See [Multi-adapter systems](#).

Passing zero is equivalent to passing one, in order to simplify the usage of single-GPU adapters.

VisibleNodeMask

For multi-adapter operation, this indicates the set of nodes where the resource is visible.

VisibleNodeMask must have the same bit set that is set in *CreationNodeMask*.

VisibleNodeMask can also have additional bits set for cross-node resources, but doing so can potentially reduce performance for resource accesses, so you should do so only when needed.

Passing zero is equivalent to passing one, in order to simplify the usage of single-GPU adapters.

Remarks

This structure is used by the following:

- [D3D12_HEAP_DESC](#) structure
- [ID3D12Resource::GetHeapProperties](#)
- [ID3D12Device::GetCustomHeapProperties](#)
- [ID3D12Device::CreateCommittedResource](#)

Valid combinations of struct member values:

- When **Type** is [D3D12_HEAP_TYPE_CUSTOM](#), **CPUPageProperty** and **MemoryPoolPreference** must not be ..._UNKNOWN.
- When **Type** is not [D3D12_HEAP_TYPE_CUSTOM](#), **CPUPageProperty** and **MemoryPoolPreference** must be ..._UNKNOWN.
- When using [D3D12_HEAP_TYPE_CUSTOM](#) and [D3D12_MEMORY_POOL_L1](#), on NUMA adapters, **CPUPageProperty** must be [D3D12_CPU_PAGE_PROPERTY_NOT_AVAILABLE](#). To differentiate NUMA from UMA adapters, see [D3D12_FEATURE_ARCHITECTURE](#) and [D3D12_FEATURE_DATA_ARCHITECTURE](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_HEAP_PROPERTIES](#)

[Core structures](#)

[Descriptor heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HEAP_SERIALIZATION_TIER enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify heap serialization support.

Syntax

C++

```
typedef enum D3D12_HEAP_SERIALIZATION_TIER {
    D3D12_HEAP_SERIALIZATION_TIER_0,
    D3D12_HEAP_SERIALIZATION_TIER_10
};
```

Constants

[] Expand table

<code>D3D12_HEAP_SERIALIZATION_TIER_0</code>
Value: (0)
Indicates that heap serialization is not supported.
<code>D3D12_HEAP_SERIALIZATION_TIER_10</code>
Value: (10)
Indicates that heap serialization is supported. Your application can serialize resource data in heaps through copying APIs such as CopyResource , without necessarily requiring an explicit state transition of resources on those heaps.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HEAP_TYPE enumeration (d3d12.h)

Article 02/14/2023

Specifies the type of heap. When resident, heaps reside in a particular physical memory pool with certain CPU cache properties.

Syntax

C++

```
typedef enum D3D12_HEAP_TYPE {
    D3D12_HEAP_TYPE_DEFAULT = 1,
    D3D12_HEAP_TYPE_UPLOAD = 2,
    D3D12_HEAP_TYPE_READBACK = 3,
    D3D12_HEAP_TYPE_CUSTOM = 4,
    D3D12_HEAP_TYPE_GPU_UPLOAD
} ;
```

Constants

[+] Expand table

D3D12_HEAP_TYPE_DEFAULT

Value: 1

Specifies the default heap. This heap type experiences the most bandwidth for the GPU, but cannot provide CPU access. The GPU can read and write to the memory from this pool, and resource transition barriers may be changed. The majority of heaps and resources are expected to be located here, and are typically populated through resources in upload heaps.

D3D12_HEAP_TYPE_UPLOAD

Value: 2

Specifies a heap used for uploading. This heap type has CPU access optimized for uploading to the GPU, but does not experience the maximum amount of bandwidth for the GPU. This heap type is best for CPU-write-once, GPU-read-once data; but GPU-read-once is stricter than necessary. GPU-read-once-or-from-cache is an acceptable use-case for the data; but such usages are hard to judge due to differing GPU cache designs and sizes. If in doubt, stick to the GPU-read-once definition or profile the difference on many GPUs between copying the data to a _DEFAULT heap vs. reading the data from an _UPLOAD heap.

Resources in this heap must be created with [D3D12_RESOURCE_STATE_GENERIC_READ](#) and

cannot be changed away from this. The CPU address for such heaps is commonly not efficient for CPU reads.

The following are typical usages for _UPLOAD heaps:

- Initializing resources in a _DEFAULT heap with data from the CPU.
- Uploading dynamic data in a constant buffer that is read, repeatedly, by each vertex or pixel.

The following are likely not good usages for _UPLOAD heaps:

- Re-initializing the contents of a resource every frame.
- Uploading constant data which is only used every other Draw call, where each Draw uses a non-trivial amount of other data.

D3D12_HEAP_TYPE_READBACK

Value: 3

Specifies a heap used for reading back. This heap type has CPU access optimized for reading data back from the GPU, but does not experience the maximum amount of bandwidth for the GPU.

This heap type is best for GPU-write-once, CPU-readable data. The CPU cache behavior is write-back, which is conducive for multiple sub-cache-line CPU reads.

Resources in this heap must be created with [D3D12_RESOURCE_STATE_COPY_DEST](#), and cannot be changed away from this.

D3D12_HEAP_TYPE_CUSTOM

Value: 4

Specifies a custom heap. The application may specify the memory pool and CPU cache properties directly, which can be useful for UMA optimizations, multi-engine, multi-adapter, or other special cases. To do so, the application is expected to understand the adapter architecture to make the right choice. For more details, see

[D3D12_FEATURE_ARCHITECTURE](#),
[D3D12_FEATURE_DATA_ARCHITECTURE](#), and
[GetCustomHeapProperties](#).

Remarks

This enum is used by the following API items:

- [D3D12_HEAP_DESC](#)
- [D3D12_HEAP_PROPERTIES](#)
- [GetCustomHeapProperties](#)

The heap types fall into two categories: abstracted heap types, and custom heap types.

The following are abstracted heap types:

- D3D12_HEAP_TYPE_DEFAULT
- D3D12_HEAP_TYPE_UPLOAD
- D3D12_HEAP_TYPE_READBACK

The following is a custom heap type:

- D3D12_HEAP_TYPE_CUSTOM

The abstracted heap types (_DEFAULT, _UPLOAD, and _READBACK) are useful to simplify writing adapter-neutral applications, because such applications don't need to be aware of the adapter memory architecture. To use an abstracted heap type to simplify writing adapter-neutral applications, the application essentially treats the adapter as if it were a discrete or NUMA adapter. But, using the heap types enables efficient translation for UMA adapters. Adapter architecture neutral applications should assume there are two memory pools available, where the pool with the most GPU bandwidth cannot provide CPU access. The pool with the least GPU bandwidth can have CPU access; but must be either optimized for upload to GPU or readback from GPU.

Note that textures (unlike buffers) can't be heap type UPLOAD or READBACK.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HIT_GROUP_DESC structure (d3d12.h)

Article 02/22/2024

Describes a raytracing hit group state subobject that can be included in a state object.

Syntax

C++

```
typedef struct D3D12_HIT_GROUP_DESC {
    LPCWSTR             HitGroupExport;
    D3D12_HIT_GROUP_TYPE Type;
    LPCWSTR             AnyHitShaderImport;
    LPCWSTR             ClosestHitShaderImport;
    LPCWSTR             IntersectionShaderImport;
} D3D12_HIT_GROUP_DESC;
```

Members

`HitGroupExport`

The name of the hit group.

`Type`

A value from the `D3D12_HIT_GROUP_TYPE` enumeration specifying the type of the hit group.

`AnyHitShaderImport`

Optional name of the any-hit shader associated with the hit group. This field can be used with all hit group types.

`ClosestHitShaderImport`

Optional name of the closest-hit shader associated with the hit group. This field can be used with all hit group types.

`IntersectionShaderImport`

Optional name of the intersection shader associated with the hit group. This field can only be used with hit groups of type procedural primitive.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HIT_GROUP_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of a raytracing hit group state subobject. Use a value from this enumeration with the [D3D12_HIT_GROUP_DESC](#) structure.

Syntax

C++

```
typedef enum D3D12_HIT_GROUP_TYPE {
    D3D12_HIT_GROUP_TYPE_TRIANGLES = 0,
    D3D12_HIT_GROUP_TYPE_PROCEDURAL_PRIMITIVE = 0x1
};
```

Constants

[+] [Expand table](#)

`D3D12_HIT_GROUP_TYPE_TRIANGLES`

Value: `0`

The hit group uses a list of triangles to calculate ray hits. Hit groups that use triangles can't contain an intersection shader.

`D3D12_HIT_GROUP_TYPE_PROCEDURAL_PRIMITIVE`

Value: `0x1`

The hit group uses a procedural primitive within a bounding box to calculate ray hits. Hit groups that use procedural primitives must contain an intersection shader.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_HIT_KIND enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_HIT_KIND {
    D3D12_HIT_KIND_TRIANGLE_FRONT_FACE,
    D3D12_HIT_KIND_TRIANGLE_BACK_FACE
};
```

Constants

[+] Expand table

D3D12_HIT_KIND_TRIANGLE_FRONT_FACE
D3D12_HIT_KIND_TRIANGLE_BACK_FACE

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_INDEX_BUFFER_STRIP_CUT_VALUE enumeration (d3d12.h)

Article 02/22/2024

When using triangle strip primitive topology, vertex positions are interpreted as vertices of a continuous triangle “strip”. There is a special index value that represents the desire to have a discontinuity in the strip, the cut index value. This enum lists the supported cut values.

Syntax

C++

```
typedef enum D3D12_INDEX_BUFFER_STRIP_CUT_VALUE {
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_DISABLED = 0,
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFF = 1,
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFFFFFF = 2
};
```

Constants

[] Expand table

<code>D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_DISABLED</code>
--

Value: 0

Indicates that there is no cut value.

<code>D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFF</code>
--

Value: 1

Indicates that 0xFFFF should be used as the cut value.

<code>D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFFFFFF</code>
--

Value: 2

Indicates that 0xFFFFFFFF should be used as the cut value.

Remarks

This enum is used by the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INDEX_BUFFER_VIEW structure (d3d12.h)

Article02/22/2024

Describes the index buffer to view.

Syntax

C++

```
typedef struct D3D12_INDEX_BUFFER_VIEW {
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation;
    UINT                     SizeInBytes;
    DXGI_FORMAT              Format;
} D3D12_INDEX_BUFFER_VIEW;
```

Members

BufferLocation

The GPU virtual address of the index buffer.

D3D12_GPU_VIRTUAL_ADDRESS is a `typedef'd` synonym of `UINT64`.

SizeInBytes

The size in bytes of the index buffer.

Format

A `DXGI_FORMAT`-typed value for the index-buffer format.

Remarks

This structure is passed into `ID3D12GraphicsCommandList::IASetIndexBuffer`.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INDIRECT_ARGUMENT_DESC structure (d3d12.h)

Article02/22/2024

Describes an indirect argument (an indirect parameter), for use with a command signature.

Syntax

C++

```
typedef struct D3D12_INDIRECT_ARGUMENT_DESC {
    D3D12_INDIRECT_ARGUMENT_TYPE Type;
    union {
        struct {
            UINT Slot;
        } VertexBuffer;
        struct {
            UINT RootParameterIndex;
            UINT DestOffsetIn32BitValues;
            UINT Num32BitValuesToSet;
        } Constant;
        struct {
            UINT RootParameterIndex;
        } ConstantBufferView;
        struct {
            UINT RootParameterIndex;
        } ShaderResourceView;
        struct {
            UINT RootParameterIndex;
        } UnorderedAccessView;
        struct {
            UINT RootParameterIndex;
            UINT DestOffsetIn32BitValues;
        } IncrementingConstant;
    };
} D3D12_INDIRECT_ARGUMENT_DESC;
```

Members

Type

A single [D3D12_INDIRECT_ARGUMENT_TYPE](#) enumeration constant.

VertexBuffer

`VertexBuffer.Slot`

Specifies the slot containing the vertex buffer address.

`Constant`

`Constant.RootParameterIndex`

Specifies the root index of the constant.

`Constant.DestOffsetIn32BitValues`

The offset, in 32-bit values, to set the first constant of the group. Supports multi-value constants at a given root index. Root constant entries must be sorted from smallest to largest `DestOffsetIn32BitValues`.

`Constant.Num32BitValuesToSet`

The number of 32-bit constants that are set at the given root index. Supports multi-value constants at a given root index.

`ConstantBufferView`

`ConstantBufferView.RootParameterIndex`

Specifies the root index of the CBV.

`ShaderResourceView`

`ShaderResourceView.RootParameterIndex`

Specifies the root index of the SRV.

`UnorderedAccessView`

`UnorderedAccessView.RootParameterIndex`

Specifies the root index of the UAV.

`IncrementingConstant`

`IncrementingConstant.RootParameterIndex`

`IncrementingConstant.DestOffsetIn32BitValues`

Remarks

Use this structure with the [D3D12_COMMAND_SIGNATURE_DESC](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Example Root Signatures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INDIRECT_ARGUMENT_TYPE enumeration (d3d12.h)

Article 01/31/2022

Specifies the type of the indirect parameter.

Syntax

C++

```
typedef enum D3D12_INDIRECT_ARGUMENT_TYPE {
    D3D12_INDIRECT_ARGUMENT_TYPE_DRAW = 0,
    D3D12_INDIRECT_ARGUMENT_TYPE_DRAW_INDEXED,
    D3D12_INDIRECT_ARGUMENT_TYPE_DISPATCH,
    D3D12_INDIRECT_ARGUMENT_TYPE_VERTEX_BUFFER_VIEW,
    D3D12_INDIRECT_ARGUMENT_TYPE_INDEX_BUFFER_VIEW,
    D3D12_INDIRECT_ARGUMENT_TYPE_CONSTANT,
    D3D12_INDIRECT_ARGUMENT_TYPE_CONSTANT_BUFFER_VIEW,
    D3D12_INDIRECT_ARGUMENT_TYPE_SHADER_RESOURCE_VIEW,
    D3D12_INDIRECT_ARGUMENT_TYPE_UNORDERED_ACCESS_VIEW,
    D3D12_INDIRECT_ARGUMENT_TYPE_DISPATCH_RAYS,
    D3D12_INDIRECT_ARGUMENT_TYPE_DISPATCH_MESH,
    D3D12_INDIRECT_ARGUMENT_TYPE_INCREMENTING_CONSTANT
} ;
```

Constants

[] Expand table

D3D12_INDIRECT_ARGUMENT_TYPE_DRAW Value: 0 Indicates the type is a Draw call.
D3D12_INDIRECT_ARGUMENT_TYPE_DRAW_INDEXED Indicates the type is a DrawIndexed call.
D3D12_INDIRECT_ARGUMENT_TYPE_DISPATCH Indicates the type is a Dispatch call.
D3D12_INDIRECT_ARGUMENT_TYPE_VERTEX_BUFFER_VIEW Indicates the type is a vertex buffer view.

D3D12_INDIRECT_ARGUMENT_TYPE_INDEX_BUFFER_VIEW

Indicates the type is an index buffer view.

D3D12_INDIRECT_ARGUMENT_TYPE_CONSTANT

Indicates the type is a constant.

D3D12_INDIRECT_ARGUMENT_TYPE_CONSTANT_BUFFER_VIEW

Indicates the type is a constant buffer view (CBV).

D3D12_INDIRECT_ARGUMENT_TYPE_SHADER_RESOURCE_VIEW

Indicates the type is a shader resource view (SRV).

D3D12_INDIRECT_ARGUMENT_TYPE_UNORDERED_ACCESS_VIEW

Indicates the type is an unordered access view (UAV).

Remarks

This enum is used by the [D3D12_INDIRECT_ARGUMENT_DESC](#) structure.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INPUT_CLASSIFICATION enumeration (d3d12.h)

Article02/22/2024

Identifies the type of data contained in an input slot.

Syntax

C++

```
typedef enum D3D12_INPUT_CLASSIFICATION {
    D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA = 0,
    D3D12_INPUT_CLASSIFICATION_PER_INSTANCE_DATA = 1
};
```

Constants

[] Expand table

<code>D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA</code>

Value: 0

Input data is per-vertex data.

<code>D3D12_INPUT_CLASSIFICATION_PER_INSTANCE_DATA</code>

Value: 1

Input data is per-instance data.

Remarks

Specify one of these values in the member of a [D3D12_INPUT_ELEMENT_DESC](#) structure to specify the type of data for the input element of a pipeline state object.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_INPUT_ELEMENT_DESC structure (d3d12.h)

Article 02/22/2024

Describes a single element for the input-assembler stage of the graphics pipeline.

Syntax

C++

```
typedef struct D3D12_INPUT_ELEMENT_DESC {
    LPCSTR SemanticName;
    UINT SemanticIndex;
    DXGI_FORMAT Format;
    UINT InputSlot;
    UINT AlignedByteOffset;
    D3D12_INPUT_CLASSIFICATION InputSlotClass;
    UINT InstanceDataStepRate;
} D3D12_INPUT_ELEMENT_DESC;
```

Members

`SemanticName`

The HLSL semantic associated with this element in a shader input-signature. See [HLSL Semantics](#) for more info.

`SemanticIndex`

The semantic index for the element. A semantic index modifies a semantic, with an integer index number. A semantic index is only needed in a case where there is more than one element with the same semantic. For example, a 4x4 matrix would have four components each with the semantic name `matrix`, however each of the four component would have different semantic indices (0, 1, 2, and 3).

`Format`

A `DXGI_FORMAT`-typed value that specifies the format of the element data.

`InputSlot`

An integer value that identifies the input-assembler. For more info, see [Input Slots](#). Valid values are between 0 and 15.

AlignedByteOffset

Optional. Offset, in bytes, to this element from the start of the vertex. Use D3D12_APPEND_ALIGNED_ELEMENT (0xffffffff) for convenience to define the current element directly after the previous one, including any packing if necessary.

InputSlotClass

A value that identifies the input data class for a single input slot.

InstanceDataStepRate

The number of instances to draw using the same per-instance data before advancing in the buffer by one element. This value must be 0 for an element that contains per-vertex data (the slot class is set to the D3D12_INPUT_PER_VERTEX_DATA member of [D3D12_INPUT_CLASSIFICATION](#)).

Remarks

This structure is a member of the [D3D12_INPUT_LAYOUT_DESC](#) structure. A pipeline state object contains a input-layout structure that defines one element being read from an input slot.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_INPUT_LAYOUT_DESC structure (d3d12.h)

Article 02/22/2024

Describes the input-buffer data for the input-assembler stage.

Syntax

C++

```
typedef struct D3D12_INPUT_LAYOUT_DESC {
    const D3D12_INPUT_ELEMENT_DESC *pInputElementDescs;
    UINT                           NumElements;
} D3D12_INPUT_LAYOUT_DESC;
```

Members

`pInputElementDescs`

An array of [D3D12_INPUT_ELEMENT_DESC](#) structures that describe the data types of the input-assembler stage.

`NumElements`

The number of input-data types in the array of input elements that the `pInputElementDescs` member points to.

Remarks

This structure is a member of the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_LIFETIME_STATE enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify the lifetime state of a lifetime-tracked object.

Syntax

C++

```
typedef enum D3D12_LIFETIME_STATE {
    D3D12_LIFETIME_STATE_IN_USE = 0,
    D3D12_LIFETIME_STATE_NOT_IN_USE
};
```

Constants

[+] Expand table

D3D12_LIFETIME_STATE_IN_USE Value: 0 Specifies that the lifetime-tracked object is in use.
D3D12_LIFETIME_STATE_NOT_IN_USE Specifies that the lifetime-tracked object is not in use.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_LOCAL_ROOT_SIGNATURE structure (d3d12.h)

Article02/22/2024

Defines a local root signature state subobject that will be used with associated shaders.

Syntax

C++

```
typedef struct D3D12_LOCAL_ROOT_SIGNATURE {
    ID3D12RootSignature *pLocalRootSignature;
} D3D12_LOCAL_ROOT_SIGNATURE;
```

Members

`pLocalRootSignature`

The root signature that will function as a local root signature. A state object holds a reference to this signature.

Remarks

The presence of this subobject in a state object is optional. The combination of global and/or local root signatures associated with any given shader function must define all resource bindings declared by the shader (with no overlap across global and local root signatures).

If any given function in a call graph (not counting calls across shader tables) is associated with a particular local root signature, any other functions in the graph must either be associated with the same local root signature or none, and the shader entry (the root of the call graph) must be associated with the local root signature. This is due to the fact that the set of code reachable from a given shader entry gets invoked from a shader identifier in a shader record, where a single set of local root arguments apply. Of course different shaders can use different local root signatures (or none), as their shader identifiers will be in different shader records.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_LOGIC_OP enumeration (d3d12.h)

Article 02/22/2024

Defines constants that specify logical operations to configure for a render target.

Syntax

C++

```
typedef enum D3D12_LOGIC_OP {
    D3D12_LOGIC_OP_CLEAR = 0,
    D3D12_LOGIC_OP_SET,
    D3D12_LOGIC_OP_COPY,
    D3D12_LOGIC_OP_COPY_INVERTED,
    D3D12_LOGIC_OP_NOOP,
    D3D12_LOGIC_OP_INVERT,
    D3D12_LOGIC_OP_AND,
    D3D12_LOGIC_OP_NAND,
    D3D12_LOGIC_OP_OR,
    D3D12_LOGIC_OP_NOR,
    D3D12_LOGIC_OP_XOR,
    D3D12_LOGIC_OP_EQUIV,
    D3D12_LOGIC_OP_AND_REVERSE,
    D3D12_LOGIC_OP_AND_INVERTED,
    D3D12_LOGIC_OP_OR_REVERSE,
    D3D12_LOGIC_OP_OR_INVERTED
};
```

Constants

 Expand table

`D3D12_LOGIC_OP_CLEAR`

Value: 0

Clears the render target (0).

`D3D12_LOGIC_OP_SET`

Sets the render target (1).

`D3D12_LOGIC_OP_COPY`

Copies the render target (`s` source from Pixel Shader output).

D3D12_LOGIC_OP_COPY_INVERTED

Performs an inverted-copy of the render target ($\sim s$).

D3D12_LOGIC_OP_NOOP

No operation is performed on the render target (d destination in the Render Target View).

D3D12_LOGIC_OP_INVERT

Inverts the render target ($\sim d$).

D3D12_LOGIC_OP_AND

Performs a logical AND operation on the render target ($s \& d$).

D3D12_LOGIC_OP_NAND

Performs a logical NAND operation on the render target ($\sim(s \& d)$).

D3D12_LOGIC_OP_OR

d).

Performs a logical OR operation on the render target (s

D3D12_LOGIC_OP_NOR

d)).

Performs a logical NOR operation on the render target ($\sim(s$

D3D12_LOGIC_OP_XOR

Performs a logical XOR operation on the render target ($s \wedge d$).

D3D12_LOGIC_OP_EQUIV

Performs a logical equal operation on the render target ($\sim(s \wedge d)$).

D3D12_LOGIC_OP_AND_REVERSE

Performs a logical AND and reverse operation on the render target ($s \& \sim d$).

D3D12_LOGIC_OP_AND_INVERTED

Performs a logical AND and invert operation on the render target ($\sim s \& d$).

D3D12_LOGIC_OP_OR_REVERSE

$\sim d$).

Performs a logical OR and reverse operation on the render target (s

D3D12_LOGIC_OP_OR_INVERTED

d).

Performs a logical OR and invert operation on the render target ($\sim s$

Remarks

This enum is used by the [D3D12_RENDER_TARGET_BLEND_DESC](#) structure.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MAKE_COARSE_SHADING_RATE macro (d3d12.h)

Article02/22/2024

Syntax

C++

```
void D3D12_MAKE_COARSE_SHADING_RATE(  
    x,  
    y  
) ;
```

Parameters

x

y

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

👍 Yes

👎 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MEASUREMENTS_ACTION enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify what should be done with the results of earlier workload instrumentation.

Syntax

C++

```
typedef enum D3D12_MEASUREMENTS_ACTION {
    D3D12_MEASUREMENTS_ACTION_KEEP_ALL = 0,
    D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS,
    D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS_HIGH_PRIORITY,
    D3D12_MEASUREMENTS_ACTION_DISCARD_PREVIOUS
};
```

Constants

[] Expand table

D3D12_MEASUREMENTS_ACTION_KEEP_ALL

Value: 0

The default setting. Specifies that all results should be kept.

D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS

Specifies that the driver has seen all the data that it's ever going to, so it should stop waiting for more and go ahead compiling optimized shaders.

D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS_HIGH_PRIORITY

Like **D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS**, but also specifies that your application doesn't care about glitches, so the runtime should ignore the usual idle priority rules and go ahead using as many threads as possible to get shader recompiles done fast. Available only in **Developer mode**.

D3D12_MEASUREMENTS_ACTION_DISCARD_PREVIOUS

Specifies that the optimization state should be reset; hinting that whatever has previously been measured no longer applies.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MEMCPY_DEST structure (d3d12.h)

Article 02/22/2024

Describes the destination of a memory copy operation.

Syntax

C++

```
typedef struct D3D12_MEMCPY_DEST {
    void    *pData;
    SIZE_T RowPitch;
    SIZE_T SlicePitch;
} D3D12_MEMCPY_DEST;
```

Members

`pData`

A pointer to a memory block that receives the copied data.

`RowPitch`

The row pitch, or width, or physical size, in bytes, of the subresource data.

`SlicePitch`

The slice pitch, or width, or physical size, in bytes, of the subresource data.

Remarks

This structure is used by a number of helper methods, refer to [Helper Structures and Functions for D3D12](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MEMORY_POOL enumeration (d3d12.h)

Article02/22/2024

Specifies the memory pool for the heap.

Syntax

C++

```
typedef enum D3D12_MEMORY_POOL {
    D3D12_MEMORY_POOL_UNKNOWN = 0,
    D3D12_MEMORY_POOL_L0 = 1,
    D3D12_MEMORY_POOL_L1 = 2
};
```

Constants

 Expand table

D3D12_MEMORY_POOL_UNKNOWN

Value: 0

The memory pool is unknown.

D3D12_MEMORY_POOL_L0

Value: 1

The memory pool is L0.

L0 is the physical system memory pool.

When the adapter is discrete/NUMA, this pool has greater bandwidth for the CPU and less bandwidth for the GPU.

When the adapter is UMA, this pool is the only one which is valid.

D3D12_MEMORY_POOL_L1

Value: 2

The memory pool is L1.

L1 is typically known as the physical video memory pool.

L1 is only available when the adapter is discrete/NUMA, and has greater bandwidth for the GPU and cannot even be accessed by the CPU.

When the adapter is UMA, this pool is not available.

Remarks

This enum is used by the [D3D12_HEAP_PROPERTIES](#) structure.

When the adapter is UMA, D3D12_MEMORY_POOL_L0 and DXGI_MEMORY_SEGMENT_GROUP_LOCAL refer to the same memory.

When

the adapter is not UMA: D3D12_MEMORY_POOL_L0 and DXGI_MEMORY_SEGMENT_GROUP_NON_LOCAL refer to the same memory.

D3D12_MEMORY_POOL_L1 and DXGI_MEMORY_SEGMENT_GROUP_LOCAL refer to the same memory.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Descriptor Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MESH_SHADER_TIER enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify mesh and amplification shader support.

Syntax

C++

```
typedef enum D3D12_MESH_SHADER_TIER {
    D3D12_MESH_SHADER_TIER_NOT_SUPPORTED = 0,
    D3D12_MESH_SHADER_TIER_1 = 10
};
```

Constants

 Expand table

<code>D3D12_MESH_SHADER_TIER_NOT_SUPPORTED</code>	Value: 0 Specifies that mesh and amplification shaders are not supported.
<code>D3D12_MESH_SHADER_TIER_1</code>	Value: 10 Specifies that mesh and amplification shaders are supported.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Mesh shader spec ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_META_COMMAND_DESC structure (d3d12.h)

Article 02/22/2024

Describes a meta command.

Syntax

C++

```
typedef struct D3D12_META_COMMAND_DESC {
    GUID Id;
    LPCWSTR Name;
    D3D12_GRAPHICS_STATES InitializationDirtyState;
    D3D12_GRAPHICS_STATES ExecutionDirtyState;
} D3D12_META_COMMAND_DESC;
```

Members

Id

Type: [GUID](#)

A [GUID](#) uniquely identifying the meta command.

Name

Type: [LPCWSTR](#)

The meta command name.

InitializationDirtyState

Type: [D3D12_GRAPHICS_STATES](#)

Declares the command list states that are modified by the call to initialize the meta command. If all state bits are set, then that's equivalent to calling [ID3D12GraphicsCommandList::ClearState](#).

ExecutionDirtyState

Type: [D3D12_GRAPHICS_STATES](#)

Declares the command list states that are modified by the call to execute the meta command. If all state bits are set, then that's equivalent to calling [ID3D12GraphicsCommandList::ClearState](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_META_COMMAND_PARAMETER_DESC structure (d3d12.h)

Article 02/22/2024

Describes a parameter to a meta command.

Syntax

C++

```
typedef struct D3D12_META_COMMAND_PARAMETER_DESC {
    LPCWSTR Name;
    D3D12_META_COMMAND_PARAMETER_TYPE Type;
    D3D12_META_COMMAND_PARAMETER_FLAGS Flags;
    D3D12_RESOURCE_STATES RequiredResourceState;
    UINT StructureOffset;
} D3D12_META_COMMAND_PARAMETER_DESC;
```

Members

Name

Type: [LPCWSTR](#)

The parameter name.

Type

Type: [D3D12_META_COMMAND_PARAMETER_TYPE](#)

A [D3D12_META_COMMAND_PARAMETER_TYPE](#) specifying the parameter type.

Flags

Type: [D3D12_META_COMMAND_PARAMETER_FLAGS](#)

A [D3D12_META_COMMAND_PARAMETER_FLAGS](#) specifying the parameter flags.

RequiredResourceState

Type: [D3D12_RESOURCE_STATES](#)

A [D3D12_RESOURCE_STATES](#) specifying the expected state of a resource parameter.

StructureOffset

Type: **UINT**

The 4-byte aligned offset for this parameter, within the structure containing the parameter values, which you pass when creating/initializing/executing the meta command, as appropriate.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_META_COMMAND_PARAMETER_FLAGS enumeration (d3d12.h)

Article 02/22/2024

Defines constants that specify the flags for a parameter to a meta command. Values can be bitwise OR'd together.

Syntax

C++

```
typedef enum D3D12_META_COMMAND_PARAMETER_FLAGS {
    D3D12_META_COMMAND_PARAMETER_FLAG_INPUT = 0x1,
    D3D12_META_COMMAND_PARAMETER_FLAG_OUTPUT = 0x2
};
```

Constants

[] Expand table

<code>D3D12_META_COMMAND_PARAMETER_FLAG_INPUT</code>	Value: <code>0x1</code> Specifies that the parameter is an input resource.
<code>D3D12_META_COMMAND_PARAMETER_FLAG_OUTPUT</code>	Value: <code>0x2</code> Specifies that the parameter is an output resource.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_META_COMMAND_PARAMETER_STAGE enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify the stage of a parameter to a meta command.

Syntax

C++

```
typedef enum D3D12_META_COMMAND_PARAMETER_STAGE {
    D3D12_META_COMMAND_PARAMETER_STAGE_CREATION = 0,
    D3D12_META_COMMAND_PARAMETER_STAGE_INITIALIZATION = 1,
    D3D12_META_COMMAND_PARAMETER_STAGE_EXECUTION = 2
};
```

Constants

[+] Expand table

	<code>D3D12_META_COMMAND_PARAMETER_STAGE_CREATION</code>
Value:	0
	Specifies that the parameter is used at the meta command creation stage.
	<code>D3D12_META_COMMAND_PARAMETER_STAGE_INITIALIZATION</code>
Value:	1
	Specifies that the parameter is used at the meta command initialization stage.
	<code>D3D12_META_COMMAND_PARAMETER_STAGE_EXECUTION</code>
Value:	2
	Specifies that the parameter is used at the meta command execution stage.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_META_COMMAND_PARAMETER_TYPE enumeration (d3d12.h)

Article 02/22/2024

Defines constants that specify the data type of a parameter to a meta command.

Syntax

C++

```
typedef enum D3D12_META_COMMAND_PARAMETER_TYPE {
    D3D12_META_COMMAND_PARAMETER_TYPE_FLOAT = 0,
    D3D12_META_COMMAND_PARAMETER_TYPE_UINT64 = 1,
    D3D12_META_COMMAND_PARAMETER_TYPE_GPU_VIRTUAL_ADDRESS = 2,
    D3D12_META_COMMAND_PARAMETER_TYPE_CPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UA
    V = 3,
    D3D12_META_COMMAND_PARAMETER_TYPE_GPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UA
    V = 4
};
```

Constants

[Expand table](#)

D3D12_META_COMMAND_PARAMETER_TYPE_FLOAT

Value: 0

Specifies that the parameter is of type [FLOAT](#).

D3D12_META_COMMAND_PARAMETER_TYPE_UINT64

Value: 1

Specifies that the parameter is of type [UINT64](#).

D3D12_META_COMMAND_PARAMETER_TYPE_GPU_VIRTUAL_ADDRESS

Value: 2

Specifies that the parameter is a GPU virtual address.

D3D12_META_COMMAND_PARAMETER_TYPE_CPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UAV

Value: 3

Specifies that the parameter is a CPU descriptor handle to a heap containing either constant buffer views, shader resource views, or unordered access views.

D3D12_META_COMMAND_PARAMETER_TYPE_GPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UAV

Value: 4

Specifies that the parameter is a GPU descriptor handle to a heap containing either constant buffer views, shader resource views, or unordered access views.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_MIP_REGION structure (d3d12.h)

Article 02/22/2024

Describes the dimensions of a mip region.

Syntax

C++

```
typedef struct D3D12_MIP_REGION {
    UINT Width;
    UINT Height;
    UINT Depth;
} D3D12_MIP_REGION;
```

Members

Width

The width of the mip region.

Height

The height of the mip region.

Depth

The depth of the mip region.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Sampler feedback specification ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MULTIPLE_FENCE_WAIT_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies multiple wait flags for multiple fences.

Syntax

C++

```
typedef enum D3D12_MULTIPLE_FENCE_WAIT_FLAGS {
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_NONE = 0,
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_ANY = 0x1,
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_ALL = 0
} ;
```

Constants

[+] Expand table

<code>D3D12_MULTIPLE_FENCE_WAIT_FLAG_NONE</code>
--

Value: *0*

No flags are being passed. This means to use the default behavior, which is to wait for all fences before signaling the event.

<code>D3D12_MULTIPLE_FENCE_WAIT_FLAG_ANY</code>

Value: *0x1*

Modifies behavior to indicate that the event should be signaled after any one of the fence values has been reached by its corresponding fence.

<code>D3D12_MULTIPLE_FENCE_WAIT_FLAG_ALL</code>

Value: *0*

An alias for `D3D12_MULTIPLE_FENCE_WAIT_FLAG_NONE`, meaning to use the default behavior and wait for all fences.

Remarks

This enum is used by the [SetEventOnMultipleFenceCompletion](#) method.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies options for determining quality levels.

Syntax

C++

```
typedef enum D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS {
    D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_NONE = 0,
    D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_TILED_RESOURCE = 0x1
};
```

Constants

[+] Expand table

<code>D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_NONE</code>

Value: 0

No options are supported.

<code>D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_TILED_RESOURCE</code>

Value: 0x1

The number of quality levels can be determined for tiled resources.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_NODE_MASK structure (d3d12.h)

Article 02/22/2024

A state subobject that identifies the GPU nodes to which the state object applies.

Syntax

C++

```
typedef struct D3D12_NODE_MASK {
    UINT NodeMask;
} D3D12_NODE_MASK;
```

Members

NodeMask

The node mask.

Remarks

This subobject is optional. In its absence, the state object applies to all available nodes. If a node mask subobject has been associated with any part of a state object, a node mask association must be made to all exports in a state object (including imported collections) and all node mask subobjects that are referenced must have matching content.

ⓘ Important

On some versions of the DirectX Runtime, specifying a node via `D3D12_NODE_MASK` in a [D3D12 STATE SUBOBJECT](#) with type [D3D12 STATE SUBOBJECT TYPE NODE MASK](#), the runtime will incorrectly handle a node mask value of `0`, which should use node #1, which will lead to errors when attempting to use the state object later. Specify an explicit node value of 1, or omit the `D3D12_NODE_MASK` subobject to avoid this issue.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_PACKED_MIP_INFO structure (d3d12.h)

Article 02/22/2024

Describes the tile structure of a tiled resource with mipmaps.

Syntax

C++

```
typedef struct D3D12_PACKED_MIP_INFO {
    UINT8 NumStandardMips;
    UINT8 NumPackedMips;
    UINT  NumTilesForPackedMips;
    UINT  StartTileIndexInOverallResource;
} D3D12_PACKED_MIP_INFO;
```

Members

NumStandardMips

The number of standard mipmaps in the tiled resource.

NumPackedMips

The number of packed mipmaps in the tiled resource.

This number starts from the least detailed mipmap (either sharing tiles or using non standard tile layout). This number is 0 if no such packing is in the resource. For array surfaces, this value is the number of mipmaps that are packed for a given array slice where each array slice repeats the same packing.

On Tier_2 tiled resources hardware, mipmaps that fill at least one standard shaped tile in all dimensions are not allowed to be included in the set of packed mipmaps. On Tier_1 hardware, mipmaps that are an integer multiple of one standard shaped tile in all dimensions are not allowed to be included in the set of packed mipmaps. Mipmaps with at least one dimension less than the standard tile shape may or may not be packed. When a given mipmap needs to be packed, all coarser mipmaps for a given array slice are considered packed as well.

NumTilesForPackedMips

The number of tiles for the packed mipmaps in the tiled resource.

If there is no packing, this value is meaningless and is set to 0. Otherwise, it is set to the number of tiles that are needed to represent the set of packed mipmaps. The pixel layout within the packed mipmaps is hardware specific. If apps define only partial mappings for the set of tiles in packed mipmaps, read and write behavior is vendor specific and undefined. For arrays, this value is only the count of packed mipmaps within the subresources for each array slice.

`StartTileIndexInOverallResource`

The offset of the first packed tile for the resource in the overall range of tiles. If **NumPackedMips** is 0, this value is meaningless and is 0. Otherwise, it is the offset of the first packed tile for the resource in the overall range of tiles for the resource. A value of 0 for **StartTileIndexInOverallResource** means the entire resource is packed. For array surfaces, this is the offset for the tiles that contain the packed mipmaps for the first array slice. Packed mipmaps for each array slice in arrayed surfaces are at this offset past the beginning of the tiles for each array slice.

Note The number of overall tiles, packed or not, for a given array slice is simply the total number of tiles for the resource divided by the resource's array size, so it is easy to locate the range of tiles for any given array slice, out of which **StartTileIndexInOverallResource** identifies which of those are packed.

Remarks

This structure is used by the [GetResourceTiling](#) method.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_PACKED_MIP_INFO](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PIPELINE_STATE_FLAGS enumeration (d3d12.h)

Article 05/29/2024

Flags to control pipeline state.

Syntax

C++

```
typedef enum D3D12_PIPELINE_STATE_FLAGS {
    D3D12_PIPELINE_STATE_FLAG_NONE = 0,
    D3D12_PIPELINE_STATE_FLAG_TOOL_DEBUG = 0x1,
    D3D12_PIPELINE_STATE_FLAG_DYNAMIC_DEPTH_BIAS,
    D3D12_PIPELINE_STATE_FLAG_DYNAMIC_INDEX_BUFFER_STRIP_CUT
} ;
```

Constants

[Expand table](#)

`D3D12_PIPELINE_STATE_FLAG_NONE`

Value: *0*

Indicates no flags.

`D3D12_PIPELINE_STATE_FLAG_TOOL_DEBUG`

Value: *0x1*

Indicates that the pipeline state should be compiled with additional information to assist debugging.

This can only be set on WARP devices.

`D3D12_PIPELINE_STATE_FLAG_DYNAMIC_DEPTH_BIAS`

Indicates that the pipeline state can be dynamically changed after the pipeline is set by using `RSSetDepthBias`.

`D3D12_PIPELINE_STATE_FLAG_DYNAMIC_INDEX_BUFFER_STRIP_CUT`

Indicates that the pipeline state can be dynamically changed after the pipeline is set by using `IASetIndexBufferStripCutValue`.

Remarks

This enum is used by the **Flags** member of the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structures.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PIPELINE_STATE_STREAM_DESC structure (d3d12.h)

Article 10/06/2021

Describes a pipeline state stream.

Syntax

C++

```
typedef struct D3D12_PIPELINE_STATE_STREAM_DESC {
    SIZE_T SizeInBytes;
    void    *pPipelineStateSubobjectStream;
} D3D12_PIPELINE_STATE_STREAM_DESC;
```

Members

SizeInBytes

SAL: *In*

Specifies the size of the opaque data structure pointed to by the *pPipelineStateSubobjectStream* member, in bytes.

pPipelineStateSubobjectStream

SAL: *In_reads(Inexpressible("Dependentonsizeofsubobjects"))*

Specifies the address of a data structure that describes as a bytestream an arbitrary pipeline state subobject.

Remarks

Use this structure with the [ID3D12Device2::CreatePipelineState](#) method to create pipeline state objects.

The format of the provided stream should consist of an alternating set of [D3D12_PIPELINE_STATE_SUBOBJECT_TYPE](#), and the corresponding subobject types for them (for example, [D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER](#) pairs with [D3D12_RASTERIZER_DESC](#)). In terms of alignment, the D3D12 runtime expects

subobjects to be individual struct pairs of enum-struct, rather than a continuous set of fields. It also expects them to be aligned to the natural word alignment of the system. This can be achieved either using `, or making a union of the enum + subobject and a void*.`

ⓘ Important

It isn't sufficient to simply union the `D3D12_PIPELINE_STATE_SUBOBJECT_TYPE` with a `void*`, because this will result in certain subobjects being misaligned. For example, `D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PRIMITIVE_TOPOLOGY` is followed by a [D3D12_PRIMITIVE_TOPOLOGY_TYPE](#) enum. If the subobject type is unioned with a `void*`, then there will be additional padding between these 2 members, resulting in corruption of the stream. Because of this, you should union the entire subobject struct with a `void*`, when `is not available`

An example of a suitable subobject for use with `D3D12_RASTERIZER_DESC` is shown here:

C++

```
struct alignas(void*) StreamingRasterizerDesc
{
private:
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE Type =
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER;
public:
    D3D12_RASTERIZER_DESC Desc;
}
```

The runtime will determine the type of a pipeline stream (valid types being **COMPUTE**, **GRAPHICS**, and **MESH**), by which subobject type, out of **VS** (vertex shader), **CS** (compute shader), and **MS** (mesh shader), is found. If the runtime finds none of these shaders, it will fail pipeline creation. If it finds multiple of these shaders which are not null, it will also fail. This means it is legal to have both, for example, a **CS** and **VS** in your stream object, provided only one has a non-null pointer for the shader bytecode for any given call to [ID3D12Device2::CreatePipelineState](#). Subobject types irrelevant to the pipeline (e.g a compute shader subobject in a graphics stream) will be ignored. If a subobject is not provided (excluding the above required subobjects), the runtime will provide a default value for it.

Consider using the `d3dx12.h` extensions for C++, which provide a set of helper structs for all pipeline subobjects (for example, the above struct is very similar to

`CD3DX12_PIPELINE_STATE_STREAM_RASTERIZER`). This header can be found under the [DirectX-Headers](#) repo on github.

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE enumeration (d3d12.h)

Article 02/14/2023

Specifies the type of a sub-object in a pipeline state stream description.

Syntax

C++

```
typedef enum D3D12_PIPELINE_STATE_SUBOBJECT_TYPE {
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_ROOT_SIGNATURE = 0,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_HS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_GS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_STREAM_OUTPUT,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_BLEND,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_MASK,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_INPUT_LAYOUT,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_IB_STRIP_CUT_VALUE,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PRIMITIVE_TOPOLOGY,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RENDER_TARGET_FORMATS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL_FORMAT,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_DESC,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_NODE_MASK,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CACHED_PSO,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_FLAGS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL1,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VIEW_INSTANCING,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_AS = 24,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MS = 25,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL2,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER1,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER2,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MAX_VALID
} ;
```

Constants

[] Expand table

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_ROOT_SIGNATURE`

Value: 0

Indicates a root signature subobject type.

The corresponding subobject type is [ID3D12RootSignature](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VS`

Indicates a vertex shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PS`

Indicates a pixel shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DS`

Indicates a domain shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_HS`

Indicates a hull shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_GS`

Indicates a geometry shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CS`

Indicates a compute shader subobject type.

The corresponding subobject type is [D3D12_SHADER_BYTECODE](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_STREAM_OUTPUT`

Indicates a stream-output subobject type.

The corresponding subobject type is [D3D12_STREAM_OUTPUT_DESC](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_BLEND`

Indicates a blend subobject type.

The corresponding subobject type is [D3D12_BLEND_DESC](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_MASK`

Indicates a sample mask subobject type.

The corresponding subobject type is [UINT](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER`

Indicates indicates a rasterizer subobject type.

The corresponding subobject type is [D3D12_RASTERIZER_DESC](#).

`D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL`

Indicates a depth stencil subobject type.

The corresponding subobject type is [D3D12_DEPTH_STENCIL_DESC](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_INPUT_LAYOUT

Indicates an input layout subobject type.

The corresponding subobject type is [D3D12_INPUT_LAYOUT_DESC](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_IB_STRIP_CUT_VALUE

Indicates an index buffer strip cut value subobject type.

The corresponding subobject type is [D3D12_INDEX_BUFFER_STRIP_CUT_VALUE](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PRIMITIVE_TOPOLOGY

Indicates a primitive topology subobject type.

The corresponding subobject type is [D3D12_PRIMITIVE_TOPOLOGY_TYPE](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RENDER_TARGET_FORMATS

Indicates a render target formats subobject type. The corresponding subobject type is

[D3D12_RT_FORMAT_ARRAY](#) structure, which wraps an array of render target formats along with a count of the array elements.

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL_FORMAT

Indicates a depth stencil format subobject.

The corresponding subobject type is [DXGI_FORMAT](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_DESC

Indicates a sample description subobject type.

The corresponding subobject type is [DXGI_SAMPLE_DESC](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_NODE_MASK

Indicates a node mask subobject type.

The corresponding subobject type is [D3D12_NODE_MASK](#) or [UINT](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CACHED_PSO

Indicates a cached pipeline state object subobject type.

The corresponding subobject type is [D3D12_CACHED_PIPELINE_STATE](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_FLAGS

Indicates a flags subobject type.

The corresponding subobject type is [D3D12_PIPELINE_STATE_FLAGS](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL1

Indicates an expanded depth stencil subobject type. This expansion of the depth stencil subobject supports optional depth bounds checking.

The corresponding subobject type is [D3D12_DEPTH_STENCIL_DESC1](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VIEW_INSTANCING

Indicates a view instancing subobject type.

The corresponding subobject type is [D3D12_VIEW_INSTANCING_DESC](#).

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_AS

Value: 24

<p>Indicates an amplification shader subobject type. The corresponding subobject type is D3D12_SHADER_BYTECODE.</p>
<p><code>D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MS</code> Value: 25 Indicates a mesh shader subobject type. The corresponding subobject type is D3D12_SHADER_BYTECODE.</p>
<p><code>D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MAX_VALID</code> A sentinel value that marks the exclusive upper-bound of valid values this enumeration represents.</p>

Remarks

This enum is used in the creation of pipeline state objects using the ID3D12Device1::CreatePipelineState method. The CreatePipelineState method takes a D3D12_PIPELINE_STATE_STREAM_DESC as one of its parameters, this structure in turn describes a bytestream made up of alternating D3D12_PIPELINE_STATE_SUBOBJECT_TYPE enumeration values and their corresponding subobject description structs. This bytestream description can be made a concrete type by defining a structure that has the same alternating pattern of alternating D3D12_PIPELINE_STATE_SUBOBJECT_TYPE enumeration values and their corresponding subobject description structs as members.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

See [D3D12_PIPELINE_STATE_STREAM_DESC](#) for a description of the layout and behavior of a streaming pipeline desc.

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PLACED_SUBRESOURCE_FOOTPRINT structure (d3d12.h)

Article 02/22/2024

Describes the footprint of a placed subresource, including the offset and the D3D12_SUBRESOURCE_FOOTPRINT.

Syntax

C++

```
typedef struct D3D12_PLACED_SUBRESOURCE_FOOTPRINT {
    UINT64 Offset;
    D3D12_SUBRESOURCE_FOOTPRINT Footprint;
} D3D12_PLACED_SUBRESOURCE_FOOTPRINT;
```

Members

Offset

The offset of the subresource within the parent resource, in bytes. The offset between the start of the parent resource and this subresource.

Footprint

The format, width, height, depth, and row-pitch of the subresource, as a D3D12_SUBRESOURCE_FOOTPRINT structure.

Remarks

This structure is used in the D3D12_TEXTURE_COPY_LOCATION structure, and by [ID3D12Device::GetCopyableFootprints](#).

All the data referenced by the footprint structure must fit within the bounds of the parent resource. If you use [GetCopyableFootprints](#) to fill out the structure, the *pTotalBytes* output field indicates the required size of the resource.

This structure is also used a number of helper functions (refer to [Helper Structures and Functions for D3D12](#)).

When copying textures, use this structure along with [D3D12_TEXTURE_COPY_LOCATION](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_PREDICATION_OP enumeration (d3d12.h)

Article02/22/2024

Specifies the predication operation to apply.

Syntax

C++

```
typedef enum D3D12_PREDICATION_OP {
    D3D12_PREDICATION_OP_EQUAL_ZERO = 0,
    D3D12_PREDICATION_OP_NOT_EQUAL_ZERO = 1
};
```

Constants

 Expand table

D3D12_PREDICATION_OP_EQUAL_ZERO

Value: 0

Enables predication if all 64-bits are zero.

D3D12_PREDICATION_OP_NOT_EQUAL_ZERO

Value: 1

Enables predication if at least one of the 64-bits are not zero.

Remarks

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

Predication is decoupled from queries. Predication can be set based on the value of 64-bits within a buffer.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Predication](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_PRIMITIVE_TOPOLOGY_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies how the pipeline interprets geometry or hull shader input primitives.

Syntax

C++

```
typedef enum D3D12_PRIMITIVE_TOPOLOGY_TYPE {
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_UNDEFINED = 0,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_POINT = 1,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_LINE = 2,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE = 3,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH = 4
};
```

Constants

[+] Expand table

D3D12_PRIMITIVE_TOPOLOGY_TYPE_UNDEFINED

Value: 0

The shader has not been initialized with an input primitive type.

D3D12_PRIMITIVE_TOPOLOGY_TYPE_POINT

Value: 1

Interpret the input primitive as a point.

D3D12_PRIMITIVE_TOPOLOGY_TYPE_LINE

Value: 2

Interpret the input primitive as a line.

D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE

Value: 3

Interpret the input primitive as a triangle.

D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH

Value: 4

Interpret the input primitive as a control point patch.

Remarks

This enum is used by the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER enumeration (d3d12.h)

Article02/22/2024

Specifies the level of support for programmable sample positions that's offered by the adapter.

Syntax

C++

```
typedef enum D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER {
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_NOT_SUPPORTED = 0,
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_1 = 1,
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_2 = 2
};
```

Constants

[] Expand table

Constant	Description
D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_NOT_SUPPORTED	Indicates that there's no support for programmable sample positions.
D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_1	Indicates that there's tier 1 support for programmable sample positions. In tier 1, a single sample pattern can be specified to repeat for every pixel (SetSamplePosition parameter <i>NumPixels</i> = 1) and ResolveSubResource is supported.
D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_2	Indicates that there's tier 2 support for programmable sample positions. In tier 2, four separate sample patterns can be specified for each pixel in a 2x2 grid (SetSamplePosition parameter <i>NumPixels</i> = 4) that repeats over the render-target or viewport, aligned on even coordinates .

Remarks

This enum is used by the [D3D12_FEATURE_D3D12_DATA_OPTIONS2](#) structure to indicate the level of support offered for programmable sample positions.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PROTECTED_RESOURCE_SESSION_DESC structure (d3d12.h)

Article 02/22/2024

Describes flags for a protected resource session, per adapter.

Syntax

C++

```
typedef struct D3D12_PROTECTED_RESOURCE_SESSION_DESC {
    UINT NodeMask;
    D3D12_PROTECTED_RESOURCE_SESSION_FLAGS Flags;
} D3D12_PROTECTED_RESOURCE_SESSION_DESC;
```

Members

NodeMask

Type: [UINT](#)

The node mask. For single GPU operation, set this to zero. If there are multiple GPU nodes, then set a bit to identify the node (the device's physical adapter) to which the protected session applies. Each bit in the mask corresponds to a single node. Only 1 bit may be set.

Flags

Type: [D3D12_PROTECTED_RESOURCE_SESSION_FLAGS](#)

Specifies the supported crypto sessions options.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PROTECTED_RESOURCE_SESSION_DESC1 structure (d3d12.h)

Article 02/22/2024

Describes flags and protection type for a protected resource session, per adapter.

Syntax

C++

```
typedef struct D3D12_PROTECTED_RESOURCE_SESSION_DESC1 {
    UINT NodeMask;
    D3D12_PROTECTED_RESOURCE_SESSION_FLAGS Flags;
    GUID ProtectionType;
} D3D12_PROTECTED_RESOURCE_SESSION_DESC1;
```

Members

NodeMask

Type: [UINT](#)

The node mask. For single GPU operation, set this to zero. If there are multiple GPU nodes, then set a bit to identify the node (the device's physical adapter) to which the protected session applies. Each bit in the mask corresponds to a single node. Only 1 bit may be set.

Flags

Type: [D3D12_PROTECTED_RESOURCE_SESSION_FLAGS](#)

Specifies the supported crypto sessions options.

ProtectionType

Type: [GUID](#)

The GUID that represents the protection type. Microsoft defines [D3D12_PROTECTED_RESOURCES_SESSION_HARDWARE_PROTECTED](#).

Using the [D3D12_PROTECTED_RESOURCES_SESSION_HARDWARE_PROTECTED](#) GUID is equivalent to calling [ID3D12Device4::CreateProtectedResourceSession](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_PROTECTED_RESOURCE_SESSION_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify protected resource session flags. These flags can be bitwise OR'd together to specify multiple flags at once.

Syntax

C++

```
typedef enum D3D12_PROTECTED_RESOURCE_SESSION_FLAGS {
    D3D12_PROTECTED_RESOURCE_SESSION_FLAG_NONE = 0
} ;
```

Constants

 Expand table

D3D12_PROTECTED_RESOURCE_SESSION_FLAG_NONE

Value: 0

Specifies no flag.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify protected resource session support.

Syntax

C++

```
typedef enum D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAGS {
    D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAG_NONE,
    D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAG_SUPPORTED
};
```

Constants

[] Expand table

<code>D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAG_NONE</code>	Value: <i>(0x0)</i> Indicates that protected resource sessions are not supported.
<code>D3D12_PROTECTED_RESOURCE_SESSION_SUPPORT_FLAG_SUPPORTED</code>	Value: <i>(0x1)</i> Indicates that protected resource sessions are supported.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_PROTECTED_SESSION_STATUS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify protected session status.

Syntax

C++

```
typedef enum D3D12_PROTECTED_SESSION_STATUS {
    D3D12_PROTECTED_SESSION_STATUS_OK = 0,
    D3D12_PROTECTED_SESSION_STATUS_INVALID = 1
};
```

Constants

[+] Expand table

	<code>D3D12_PROTECTED_SESSION_STATUS_OK</code>
Value:	0
	Indicates that the protected session is in a valid state.
	<code>D3D12_PROTECTED_SESSION_STATUS_INVALID</code>
Value:	1
	Indicates that the protected session is not in a valid state.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_QUERY_DATA_PIPELINE_STATISTICS structure (d3d12.h)

Article04/02/2021

Query information about graphics-pipeline activity in between calls to [BeginQuery](#) and [EndQuery](#).

Syntax

C++

```
typedef struct D3D12_QUERY_DATA_PIPELINE_STATISTICS {
    UINT64 IAVertices;
    UINT64 IAPrimitives;
    UINT64 VSInvocations;
    UINT64 GSInvocations;
    UINT64 GSPrimitives;
    UINT64 CInvocations;
    UINT64 CPrimitives;
    UINT64 PSInvocations;
    UINT64 HSInvocations;
    UINT64 DSInvocations;
    UINT64 CSInvocations;
} D3D12_QUERY_DATA_PIPELINE_STATISTICS;
```

Members

`IAVertices`

Number of vertices read by input assembler.

`IAPrimitives`

Number of primitives read by the input assembler. This number can be different depending on the primitive topology used. For example, a triangle strip with 6 vertices will produce 4 triangles, however a triangle list with 6 vertices will produce 2 triangles.

`VSI invocations`

Specifies the number of vertex shader invocations. Direct3D invokes the vertex shader once per vertex.

`GSInvocations`

Specifies the number of geometry shader invocations. When the geometry shader is set to NULL, this statistic may or may not increment depending on the graphics adapter.

GSPrimitives

Specifies the number of geometry shader output primitives.

CInvocations

Number of primitives that were sent to the rasterizer. When the rasterizer is disabled, this will not increment.

CPrimitives

Number of primitives that were rendered. This may be larger or smaller than CInvocations because after a primitive is clipped sometimes it is either broken up into more than one primitive or completely culled.

PSInvocations

Specifies the number of pixel shader invocations.

HSInvocations

Specifies the number of hull shader invocations.

DSInvocations

Specifies the number of domain shader invocations.

CSInvocations

Specifies the number of compute shader invocations.

Remarks

Use this structure with [D3D12_QUERY_HEAP_TYPE](#) and [CreateQueryHeap](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_QUERY_DATA_PIPELINE_STATISTICS1 structure (d3d12.h)

Article 02/22/2024

Syntax

C++

```
typedef struct D3D12_QUERY_DATA_PIPELINE_STATISTICS1 {
    UINT64 IAVertices;
    UINT64 IAPrimitives;
    UINT64 VSInvocations;
    UINT64 GSInvocations;
    UINT64 GSPrimitives;
    UINT64 CInvocations;
    UINT64 CPrimitives;
    UINT64 PSInvocations;
    UINT64 HSInvocations;
    UINT64 DSInvocations;
    UINT64 CSInvocations;
    UINT64 ASInvocations;
    UINT64 MSInvocations;
    UINT64 MSPrimitives;
} D3D12_QUERY_DATA_PIPELINE_STATISTICS1;
```

Members

IAVertices

IAPrimitives

VSInvocations

GSInvocations

GSPrimitives

CInvocations

CPrimitives

PSInvocations

HSInvocations

DSInvocations

CSInvocations

ASInvocations

MSInvocations

MSPrimitives

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_QUERY_DATA_SO_STATISTICS structure (d3d12.h)

Article 02/22/2024

Describes query data about the amount of data streamed out to the stream-output buffers.

Syntax

C++

```
typedef struct D3D12_QUERY_DATA_SO_STATISTICS {
    UINT64 NumPrimitivesWritten;
    UINT64 PrimitivesStorageNeeded;
} D3D12_QUERY_DATA_SO_STATISTICS;
```

Members

`NumPrimitivesWritten`

Type: [UINT64](#)

The number of primitives (that is, points, lines, and triangles) that were actually written to the stream output resource.

`PrimitivesStorageNeeded`

Type: [UINT64](#)

If the stream output resource is large enough, then *PrimitivesStorageNeeded* represents the total number of primitives written to the stream output resource. Otherwise, it represents the total number of primitives that *would* have been written to the stream-output resource had there been enough space for them all.

Remarks

Use this structure with [D3D12_QUERY_HEAP_TYPE](#) and [CreateQueryHeap](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

- [Core structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_QUERY_HEAP_DESC structure (d3d12.h)

Article 02/22/2024

Describes the purpose of a query heap. A query heap contains an array of individual queries.

Syntax

C++

```
typedef struct D3D12_QUERY_HEAP_DESC {
    D3D12_QUERY_HEAP_TYPE Type;
    UINT                 Count;
    UINT                 NodeMask;
} D3D12_QUERY_HEAP_DESC;
```

Members

Type

Specifies one member of [D3D12_QUERY_HEAP_TYPE](#).

Count

Specifies the number of queries the heap should contain.

NodeMask

For single GPU operation, set this to zero. If there are multiple GPU nodes, set a bit to identify the node (the device's physical adapter) to which the query heap applies. Each bit in the mask corresponds to a single node. Only 1 bit must be set. Refer to [Multi-adapter systems](#).

Remarks

Use this structure with [CreateQueryHeap](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_QUERY_HEAP_TYPE enumeration (d3d12.h)

Article01/31/2022

Specifies the type of query heap to create.

Syntax

C++

```
typedef enum D3D12_QUERY_HEAP_TYPE {
    D3D12_QUERY_HEAP_TYPE_OCCLUSION = 0,
    D3D12_QUERY_HEAP_TYPE_TIMESTAMP = 1,
    D3D12_QUERY_HEAP_TYPE_PIPELINE_STATISTICS = 2,
    D3D12_QUERY_HEAP_TYPE_SO_STATISTICS = 3,
    D3D12_QUERY_HEAP_TYPE_VIDEO_DECODE_STATISTICS = 4,
    D3D12_QUERY_HEAP_TYPE_COPY_QUEUE_TIMESTAMP = 5,
    D3D12_QUERY_HEAP_TYPE_PIPELINE_STATISTICS1
} ;
```

Constants

[+] Expand table

D3D12_QUERY_HEAP_TYPE_OCCLUSION

Value: 0

This returns a binary 0/1 result: 0 indicates that no samples passed depth and stencil testing, 1 indicates that at least one sample passed depth and stencil testing. This enables occlusion queries to not interfere with any GPU performance optimization associated with depth/stencil testing.

D3D12_QUERY_HEAP_TYPE_TIMESTAMP

Value: 1

Indicates that the heap is for high-performance timing data.

D3D12_QUERY_HEAP_TYPE_PIPELINE_STATISTICS

Value: 2

Indicates the heap is to contain pipeline data. Refer to [D3D12_QUERY_DATA_PIPELINE_STATISTICS](#).

D3D12_QUERY_HEAP_TYPE_SO_STATISTICS

Value: 3

Indicates the heap is to contain stream output data. Refer to [D3D12_QUERY_DATA_SO_STATISTICS](#).

D3D12_QUERY_HEAP_TYPE_VIDEO_DECODE_STATISTICS

Value: 4

Indicates the heap is to contain video decode statistics data. Refer to [D3D12_QUERY_DATA_VIDEO_DECODE_STATISTICS](#).

Video decode statistics can only be queried from video decode command lists ([D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE](#)). See [D3D12_QUERY_TYPE_DECODE_STATISTICS](#) for more details.

D3D12_QUERY_HEAP_TYPE_COPY_QUEUE_TIMESTAMP

Value: 5

Indicates the heap is to contain timestamp queries emitted exclusively by copy command lists. Copy queue timestamps can only be queried from a copy command list, and a copy command list can not emit to a regular timestamp query Heap.

Support for this query heap type is not universal. You must use [CheckFeatureSupport](#) with [D3D12_FEATURE_D3D12_OPTIONS3](#) to determine whether the adapter supports copy queue timestamp queries.

Remarks

This enum is used by the [D3D12_QUERY_HEAP_DESC](#) structure.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

D3D12_QUERY_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of query.

Syntax

C++

```
typedef enum D3D12_QUERY_TYPE {
    D3D12_QUERY_TYPE_OCCLUSION = 0,
    D3D12_QUERY_TYPE_BINARY_OCCLUSION = 1,
    D3D12_QUERY_TYPE_TIMESTAMP = 2,
    D3D12_QUERY_TYPE_PIPELINE_STATISTICS = 3,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM0 = 4,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM1 = 5,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM2 = 6,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM3 = 7,
    D3D12_QUERY_TYPE_VIDEO_DECODE_STATISTICS = 8,
    D3D12_QUERY_TYPE_PIPELINE_STATISTICS1
} ;
```

Constants

[] Expand table

`D3D12_QUERY_TYPE_OCCLUSION`

Value: 0

Indicates the query is for depth/stencil occlusion counts.

`D3D12_QUERY_TYPE_BINARY_OCCLUSION`

Value: 1

Indicates the query is for a binary depth/stencil occlusion statistics.

This new query type acts like `D3D12_QUERY_TYPE_OCCLUSION` except that it returns simply a binary 0/1 result: 0 indicates that no samples passed depth and stencil testing, 1 indicates that at least one sample passed depth and stencil testing. This enables occlusion queries to not interfere with any GPU performance optimization associated with depth/stencil testing.

`D3D12_QUERY_TYPE_TIMESTAMP`

Value: 2

Indicates the query is for high definition GPU and CPU timestamps.

`D3D12_QUERY_TYPE_PIPELINE_STATISTICS`

Value: 3

Indicates the query type is for graphics pipeline statistics, refer to [D3D12_QUERY_DATA_PIPELINE_STATISTICS](#).

`D3D12_QUERY_TYPE_SO_STATISTICS_STREAM0`

Value: 4

Stream 0 output statistics. In Direct3D 12 there is no single stream output (SO) overflow query for all the output streams. Apps need to issue multiple single-stream queries, and then correlate the results. Stream output is the ability of the GPU to write vertices to a buffer. The stream output counters monitor progress.

`D3D12_QUERY_TYPE_SO_STATISTICS_STREAM1`

Value: 5

Stream 1 output statistics.

`D3D12_QUERY_TYPE_SO_STATISTICS_STREAM2`

Value: 6

Stream 2 output statistics.

`D3D12_QUERY_TYPE_SO_STATISTICS_STREAM3`

Value: 7

Stream 3 output statistics.

`D3D12_QUERY_TYPE_VIDEO_DECODE_STATISTICS`

Value: 8

Video decode statistics. Refer to [D3D12_QUERY_DATA_VIDEO_DECODE_STATISTICS](#).

Use this query type to determine if a video was successfully decoded. If decoding fails due to insufficient BitRate or FrameRate parameters set during creation of the decode heap, then the status field of the query is set to [D3D12_VIDEO_DECODE_STATUS_RATE_EXCEEDED](#) and the query also contains new BitRate and FrameRate values that would succeed.

This query type can only be performed on video decode command lists ([D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE](#)). This query type does not use `ID3D12VideoDecodeCommandList::BeginQuery`, only `ID3D12VideoDecodeCommandList::EndQuery`. Statistics are recorded only for the most recent `ID3D12VideoDecodeCommandList::DecodeFrame` call in the same command list.

Decode status structures are defined by the codec specification.

Remarks

This enum is used by `BeginQuery`, `EndQuery` and `ResolveQueryData`.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RANGE structure (d3d12.h)

Article 02/22/2024

Describes a memory range.

Syntax

C++

```
typedef struct D3D12_RANGE {
    SIZE_T Begin;
    SIZE_T End;
} D3D12_RANGE;
```

Members

Begin

The offset, in bytes, denoting the beginning of a memory range.

End

The offset, in bytes, denoting the end of a memory range. **End** is one-past-the-end.

Remarks

End is one-past-the-end. When **Begin** equals **End**, the range is empty. The size of the range is (**End** - **Begin**).

This structure is used by the [Map](#) and [Unmap](#) methods.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RANGE](#)

[Core Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RANGE_UINT64 structure (d3d12.h)

Article 02/22/2024

Describes a memory range in a 64-bit address space.

Syntax

C++

```
typedef struct D3D12_RANGE_UINT64 {
    UINT64 Begin;
    UINT64 End;
} D3D12_RANGE_UINT64;
```

Members

Begin

The offset, in bytes, denoting the beginning of a memory range.

End

The offset, in bytes, denoting the end of a memory range. **End** is one-past-the-end.

Remarks

End is one-past-the-end. When **Begin** equals **End**, the range is empty. The size of the range is (**End** - **Begin**).

This structure is used by the [D3D12_SUBRESOURCE_RANGE_UINT64](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RASTERIZER_DESC structure (d3d12.h)

Article12/06/2022

Describes rasterizer state.

Syntax

C++

```
typedef struct D3D12_RASTERIZER_DESC {
    D3D12_FILL_MODE           FillMode;
    D3D12_CULL_MODE           CullMode;
    BOOL                      FrontCounterClockwise;
    INT                       DepthBias;
    FLOAT                     DepthBiasClamp;
    FLOAT                     SlopeScaledDepthBias;
    BOOL                      DepthClipEnable;
    BOOL                      MultisampleEnable;
    BOOL                      AntialiasedLineEnable;
    UINT                      ForcedSampleCount;
    D3D12_CONSERVATIVE_RASTERIZATION_MODE ConservativeRaster;
} D3D12_RASTERIZER_DESC;
```

Members

`FillMode`

A [D3D12_FILL_MODE](#)-typed value that specifies the fill mode to use when rendering.

`CullMode`

A [D3D12_CULL_MODE](#)-typed value that specifies that triangles facing the specified direction are not drawn.

`FrontCounterClockwise`

Determines if a triangle is front- or back-facing. If this member is **TRUE**, a triangle will be considered front-facing if its vertices are counter-clockwise on the render target and considered back-facing if they are clockwise. If this parameter is **FALSE**, the opposite is true.

`DepthBias`

Depth value added to a given pixel. For info about depth bias, see [Depth Bias](#).

DepthBiasClamp

Maximum depth bias of a pixel. For info about depth bias, see [Depth Bias](#).

SlopeScaledDepthBias

Scalar on a given pixel's slope. For info about depth bias, see [Depth Bias](#).

DepthClipEnable

Specifies whether to enable clipping based on distance.

The hardware always performs x and y clipping of rasterized coordinates. When **DepthClipEnable** is set to the default—**TRUE**, the hardware also clips the z value (that is, the hardware performs the last step of the following algorithm).

syntax

```
0 < w  
-w <= x <= w (or arbitrarily wider range if implementation uses a guard band  
to reduce clipping burden)  
-w <= y <= w (or arbitrarily wider range if implementation uses a guard band  
to reduce clipping burden)  
0 <= z <= w
```

When you set **DepthClipEnable** to **FALSE**, the hardware skips the z clipping (that is, the last step in the preceding algorithm). However, the hardware still performs the "0 < w" clipping. When z clipping is disabled, improper depth ordering at the pixel level might result. However, when z clipping is disabled, stencil shadow implementations are simplified. In other words, you can avoid complex special-case handling for geometry that goes beyond the back clipping plane.

MultisampleEnable

Specifies whether to use the quadrilateral or alpha line anti-aliasing algorithm on multisample antialiasing (MSAA) render targets. Set to **TRUE** to use the quadrilateral line anti-aliasing algorithm and to **FALSE** to use the alpha line anti-aliasing algorithm. For more info about this member, see Remarks.

AntialiasedLineEnable

Specifies whether to enable line antialiasing; only applies if doing line drawing and **MultisampleEnable** is **FALSE**. For more info about this member, see Remarks.

ForcedSampleCount

Type: **UINT**

The sample count that is forced while UAV rendering or rasterizing. Valid values are 0, 1, 4, 8, and optionally 16. 0 indicates that the sample count is not forced.

Note If you want to render with **ForcedSampleCount** set to 1 or greater, you must follow these guidelines:

- Don't bind depth-stencil views.
- Disable depth testing.
- Ensure the shader doesn't output depth.
- If you have any render-target views bound ([D3D12_DESCRIPTOR_HEAP_TYPE RTV](#)) and **ForcedSampleCount** is greater than 1, ensure that every render target has only a single sample.
- Don't operate the shader at sample frequency. Therefore, [ID3D12ShaderReflection::IsSampleFrequencyShader](#) returns **FALSE**.

Otherwise, rendering behavior is undefined.

ConservativeRaster

A [D3D12_CONSERVATIVE_RASTERIZATION_MODE](#)-typed value that identifies whether conservative rasterization is on or off.

Remarks

A [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) contains a rasterizer-state structure.

Rasterizer state defines the behavior of the rasterizer stage.

If you do not specify some rasterizer state, the Direct3D runtime uses the following default values for rasterizer state.

[+] Expand table

State	Default Value
FillMode	D3D12_FILL_MODE_SOLID

CullMode	D3D12_CULL_MODE_BACK
FrontCounterClockwise	FALSE
DepthBias	0
DepthBiasClamp	0.0f
SlopeScaledDepthBias	0.0f
DepthClipEnable	TRUE
MultisampleEnable	FALSE
AntialiasedLineEnable	FALSE
ForcedSampleCount	0
ConservativeRaster	D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF

Note For [feature levels](#) 9.1, 9.2, 9.3, and 10.0, if you set **MultisampleEnable** to **FALSE**, the runtime renders all points, lines, and triangles without anti-aliasing even for render targets with a sample count greater than 1. For feature levels 10.1 and higher, the setting of **MultisampleEnable** has no effect on points and triangles with regard to MSAA and impacts only the selection of the line-rendering algorithm as shown in this table:

[+] [Expand table](#)

Line-rendering algorithm	MultisampleEnable	AntialiasedLineEnable
Aliased	FALSE	FALSE
Alpha antialiased	FALSE	TRUE
Quadrilateral	TRUE	FALSE
Quadrilateral	TRUE	TRUE

The settings of the **MultisampleEnable** and **AntialiasedLineEnable** members apply only to multisample antialiasing (MSAA) render targets (that is, render targets with sample counts greater than 1). Because of the differences in [feature-level](#) behavior and as long as you aren't performing any line drawing or don't mind that lines render as

quadrilaterals, we recommend that you always set `MultisampleEnable` to `TRUE` whenever you render on MSAA render targets.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RASTERIZER_DESC](#)

[Conservative Rasterization](#)

[Core Structures](#)

[Rasterizer Ordered Views](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAY_FLAGS enumeration (d3d12.h)

Article 01/31/2022

Flags passed to the [TraceRay](#) function to override transparency, culling, and early-out behavior.

Syntax

C++

```
typedef enum D3D12_RAY_FLAGS {
    D3D12_RAY_FLAG_NONE = 0,
    D3D12_RAY_FLAG_FORCE_OPAQUE = 0x1,
    D3D12_RAY_FLAG_FORCE_NON_OPAQUE = 0x2,
    D3D12_RAY_FLAG_ACCEPT_FIRST_HIT_AND_END_SEARCH = 0x4,
    D3D12_RAY_FLAG_SKIP_CLOSEST_HIT_SHADER = 0x8,
    D3D12_RAY_FLAG_CULL_BACK_FACING_TRIANGLES = 0x10,
    D3D12_RAY_FLAG_CULL_FRONT_FACING_TRIANGLES = 0x20,
    D3D12_RAY_FLAG_CULL_OPAQUE = 0x40,
    D3D12_RAY_FLAG_CULL_NON_OPAQUE = 0x80,
    D3D12_RAY_FLAG_SKIP_TRIANGLES,
    D3D12_RAY_FLAG_SKIP_PROCEDURAL_PRIMITIVES
} ;
```

Constants

[+] [Expand table](#)

D3D12_RAY_FLAG_NONE

Value: 0

No options selected.

D3D12_RAY_FLAG_FORCE_OPAQUE

Value: 0x1

All ray-primitive intersections encountered in a raytrace are treated as opaque. So no any hit shaders will be executed regardless of whether or not the hit geometry specifies

D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE, and regardless of the instance flags on the instance that was hit.

This flag is mutually exclusive with RAY_FLAG_FORCE_NON_OPAQUE, RAY_FLAG_CULL_OPAQUE and RAY_FLAG_CULL_NON_OPAQUE.

D3D12_RAY_FLAG_FORCE_NON_OPAQUE

Value: 0x2

All ray-primitive intersections encountered in a raytrace are treated as non-opaque. So any hit shaders, if present, will be executed regardless of whether or not the hit geometry specifies D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE, and regardless of the instance flags on the instance that was hit. This flag is mutually exclusive with RAY_FLAG_FORCE_OPAQUE, RAY_FLAG_CULL_OPAQUE and RAY_FLAG_CULL_NON_OPAQUE.

D3D12_RAY_FLAG_ACCEPT_FIRST_HIT_AND_END_SEARCH

Value: 0x4

The first ray-primitive intersection encountered in a raytrace automatically causes [AcceptHitAndEndSearch](#) to be called immediately after the any hit shader, including if there is no any hit shader.

The only exception is when the preceding any hit shader calls [IgnoreHit](#), in which case the ray continues unaffected such that the next hit becomes another candidate to be the first hit. For this exception to apply, the any hit shader has to actually be executed. So if the any hit shader is skipped because the hit is treated as opaque (e.g. due to RAY_FLAG_FORCE_OPAQUE or D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE or D3D12_RAYTRACING_INSTANCE_FLAG_OPAQUE being set), then [AcceptHitAndEndSearch](#) is called.

If a closest hit shader is present at the first hit, it gets invoked unless RAY_FLAG_SKIP_CLOSEST_HIT_SHADER is also present. The one hit that was found is considered "closest", even though other potential hits that might be closer on the ray may not have been visited.

A typical use for this flag is for shadows, where only a single hit needs to be found.

D3D12_RAY_FLAG_SKIP_CLOSEST_HIT_SHADER

Value: 0x8

Even if at least one hit has been committed, and the hit group for the closest hit contains a closest hit shader, skip execution of that shader.

D3D12_RAY_FLAG_CULL_BACK_FACING_TRIANGLES

Value: 0x10

Enables culling of back facing triangles. See [D3D12_RAYTRACING_INSTANCE_FLAGS](#) for selecting which triangles are back facing, per-instance.

On instances that specify D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE, this flag has no effect.

On geometry types other than D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES, this flag has no effect.

This flag is mutually exclusive with RAY_FLAG_CULL_FRONT_FACING_TRIANGLES.

D3D12_RAY_FLAG_CULL_FRONT_FACING_TRIANGLES

Value: 0x20

Enables culling of front facing triangles. See [D3D12_RAYTRACING_INSTANCE_FLAGS](#) for selecting which triangles are back facing, per-instance.

On instances that specify D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE, this flag has no effect.

On geometry types other than D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES, this flag has no effect.

This flag is mutually exclusive with RAY_FLAG_CULL_FRONT_FACING_TRIANGLES.

D3D12_RAY_FLAG_CULL_OPAQUE

Value: 0x40

Culls all primitives that are considered opaque based on their geometry and instance flags.

This flag is mutually exclusive with RAY_FLAG_FORCE_OPAQUE, RAY_FLAG_FORCE_NON_OPAQUE, and RAY_FLAG_CULL_NON_OPAQUE.

D3D12_RAY_FLAG_CULL_NON_OPAQUE

Value: 0x80

Culls all primitives that are considered non-opaque based on their geometry and instance flags.

This flag is mutually exclusive with RAY_FLAG_FORCE_OPAQUE, RAY_FLAG_FORCE_NON_OPAQUE, and RAY_FLAG_CULL_OPAQUE.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_RAYTRACING_AABB structure (d3d12.h)

Article 02/22/2024

Represents an axis-aligned bounding box (AABB) used as raytracing geometry.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_AABB {
    FLOAT MinX;
    FLOAT MinY;
    FLOAT MinZ;
    FLOAT MaxX;
    FLOAT MaxY;
    FLOAT MaxZ;
} D3D12_RAYTRACING_AABB;
```

Members

MinX

The minimum X coordinate of the box.

MinY

The minimum Y coordinate of the box.

MinZ

The minimum Z coordinate of the box.

MaxX

The maximum X coordinate of the box.

MaxY

The maximum Y coordinate of the box.

MaxZ

The maximum Z coordinate of the box.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies flags for the build of a raytracing acceleration structure. Use a value from this enumeration with the [D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#) structure that provides input to the acceleration structure build operation.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_NONE = 0,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE = 0x1,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION = 0x2,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_TRACE =
        0x4,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD =
        0x8,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_MINIMIZE_MEMORY = 0x10,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE = 0x20
} ;
```

Constants

[+] Expand table

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_NONE`

Value: 0

No options specified for the acceleration structure build.

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE`

Value: 0x1

Build the acceleration structure such that it supports future updates (via the flag

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE`) instead of the app having to entirely rebuild the structure. This option may result in increased memory consumption, build times, and lower raytracing performance. Future updates, however, should be faster than building the equivalent acceleration structure from scratch.

This flag can only be set on an initial acceleration structure build, or on an update where the source acceleration structure specified

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE](#). In other words, after an acceleration structure was built without

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE](#), no other acceleration structures can be created from it via updates.

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION](#)

Value: 0x2

Enables the option to compact the acceleration structure by calling

[CopyRaytracingAccelerationStructure](#) using compact mode, specified with

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT](#).

This option may result in increased memory consumption and build times. After future compaction, however, the resulting acceleration structure should consume a smaller memory footprint than building the acceleration structure from scratch.

This flag is compatible with all other flags. If specified as part of an acceleration structure update, the source acceleration structure must have also been built with this flag. In other words, after an acceleration structure was built without

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION](#), no other acceleration structures can be created from it via updates that specify

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION](#).

Specifying ALLOW_COMPACTION may increase pre-compaction acceleration structure size versus not specifying ALLOW_COMPACTION.

If multiple incremental builds are performed before finally compacting, there may be redundant compaction related work performed.

The size required for the compacted acceleration structure can be queried before compaction via [EmitRaytracingAccelerationStructurePostbuildInfo](#). See

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC](#) for more information on properties of compacted acceleration structure size.

Note When

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDA](#)
[TE](#) is specified, there is certain information that needs to be retained in the acceleration structure, and compaction will only help so much. However, if the

pipeline knows that the acceleration structure will no longer be updated, it can make the structure more compact. Some apps may benefit from compacting twice - once after the initial build, and again after the acceleration structure has settled to a static state, if that occurs.

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_TRACE

Value: 0x4

Construct a high quality acceleration structure that maximizes raytracing performance at the expense of additional build time. Typically, the implementation will take 2-3 times the build time than the default setting in order to get better tracing performance.

This flag is recommended for static geometry in particular. It is compatible with all other flags except for **D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD**.

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD

Value: 0x8

Construct a lower quality acceleration structure, trading raytracing performance for build speed. Typically, the implementation will take 1/2 to 1/3 the build time than default setting, with a sacrifice in tracing performance.

This flag is compatible with all other flags except for

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD.

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_MINIMIZE_MEMORY

Value: 0x10

Minimize the amount of scratch memory used during the acceleration structure build as well as the size of the result. This option may result in increased build times and/or raytracing times. This is orthogonal to the

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION flag and the explicit acceleration structure compaction that it enables. Combining the flags can mean both the initial acceleration structure as well as the result of compacting it use less memory.

The impact of using this flag for a build is reflected in the result of calling [GetRaytracingAccelerationStructurePrebuildInfo](#) before doing the build to retrieve memory requirements for the build.

This flag is compatible with all other flags.

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE

Value: 0x20

Perform an acceleration structure update, as opposed to building from scratch. This is faster than a full build, but can negatively impact raytracing performance, especially if the positions of the underlying objects have changed significantly from the original build of the acceleration structure before updates.

If the addresses of the source and destination acceleration structures are identical, the update is performed in-place. Any other overlapping of address ranges of the source and destination is invalid. For non-overlapping source and destinations, the source acceleration structure is unmodified. The memory requirement for the output acceleration structure is the same as in the input acceleration structure

The source acceleration structure must have been built with
`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE`.

This flag is compatible with all other flags. The other flags selections, aside from `D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE` and `D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE`, must match the flags in the source acceleration structure.

Acceleration structure updates can be performed in unlimited succession, as long as the source acceleration structure was created with
`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE` and the flags for the update build continue to specify
`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE`.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE enumeration (d3d12.h)

Article01/31/2022

Specifies the type of copy operation performed when calling [CopyRaytracingAccelerationStructure](#).

Syntax

C++

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_CLONE = 0,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT = 0x1,

    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_VISUALIZATION_DECODE_FOR_T
    OOLS = 0x2,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE = 0x3,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_DESERIALIZE = 0x4
} ;
```

Constants

[+] [Expand table](#)

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_CLONE`

Value: 0

Copy an acceleration structure while fixing any self-referential pointers that may be present so that the destination is a self-contained copy of the source. Any external pointers to other acceleration structures remain unchanged from source to destination in the copy. The size of the destination is identical to the size of the source.

Important

The source memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#).The

destination memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#)

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT`

Value: *0x1*

Produces a functionally equivalent acceleration structure to source in the destination, similar to the clone mode, but also fits the destination into a potentially smaller, and certainly not larger, memory footprint. The size required for the destination can be retrieved beforehand from [EmitRaytracingAccelerationStructurePostbuildInfo](#).

This mode is only valid if the source acceleration structure was originally built with the [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION](#) flag, otherwise results are undefined.

Compacting geometry requires the entire acceleration structure to be constructed, which is why you must first build and then compact the structure.

Important

The source memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#).The

destination memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#).

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_VISUALIZATION_DECODE_FOR_TOOLS`

Value: *0x2*

The destination is takes the layout described in

[D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER](#). The size required for the destination can be retrieved beforehand from

[EmitRaytracingAccelerationStructurePostbuildInfo](#).

This mode is only intended for tools such as PIX, though nothing stops any app from using it. The output is essentially the inverse of an acceleration structure build. This overall structure with is sufficient for tools/PIX to be able to give the application some visual sense of the acceleration structure the driver made out of the app's input. Visualization can help reveal driver bugs in acceleration structures if what is shown grossly mismatches the data the application used to create the acceleration structure, beyond allowed tolerances.

For top-level acceleration structures, the output includes a set of instance descriptions that are identical to the data used in the original build and in the same order. For bottom-level acceleration structures, the output includes a set of geometry descriptions roughly matching the data used in the original build. The output is only a rough match for the original in part because of the tolerances allowed in the specification for acceleration structures and in part due to the inherent complexity of reporting exactly the same structure as is conceptually encoded. For

example. axis-aligned bounding boxes (AABBs) returned for procedural primitives could be more conservative (larger) in volume and even different in number than what is actually in the acceleration structure representation. Geometries, each with its own geometry description, appear in the same order as in the original acceleration, as shader table indexing calculations depend on this.

Important

The source memory must be in state

[**D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE**](#). The destination memory must be in state
[**D3D12 RESOURCE STATE UNORDERED ACCESS**](#).

This mode is only permitted when developer mode is enabled in the OS.

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE`

Value: `0x3`

Destination takes the layout and size described in the documentation for [**D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC**](#), itself a structure generated with a call to [`EmitRaytracingAccelerationStructurePostbuildInfo`](#).

This mode serializes an acceleration structure so that an app or tools can store it to a file for later reuse, typically on a different device instance, via deserialization.

When serializing a top-level acceleration structure, the bottom-level acceleration structures it refers to do not have to still be present or intact in memory. Likewise, bottom-level acceleration structures can be serialized independent of whether any top-level acceleration structures are pointing to them. In other words, the order of serialization of acceleration structures doesn't matter.

Important

The source memory must be in state

[**D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE**](#). The destination memory must be in state
[**D3D12 RESOURCE STATE UNORDERED ACCESS**](#).

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_DESERIALIZE`

Value: `0x4`

The source must be a serialized acceleration structure, with any pointers, directly after the header, fixed to point to their new locations. For more information, see

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC](#).

The destination gets an acceleration structure that is functionally equivalent to the acceleration structure that was originally serialized. It does not matter what order top-level and bottom-level acceleration structures are deserialized, as long as by the time a top-level acceleration structure is used for raytracing or acceleration structure updates the bottom-level acceleration structures it references are present.

Deserialization can only be performed on the same device and driver version on which the data was serialized. Otherwise, the results are undefined.

This mode is only intended for tools such as PIX, though nothing stops any app from using it, but this mode is only permitted when developer mode is enabled in the OS. This copy operation is not intended to be used for caching acceleration structures, because running a full acceleration structure build is likely to be faster than loading one from disk.

ⓘ Important

The source memory must be in state

[D3D12 RESOURCE STATE NON PIXEL SHADER RESOURCE](#).The destination memory must be in state

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC structure (d3d12.h)

Article02/22/2024

Describes the space requirement for acceleration structure after compaction.

Syntax

C++

```
typedef struct
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC {
    UINT64 CompactedSizeInBytes;
}
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC;
```

Members

CompactedSizeInBytes

The space requirement for acceleration structure after compaction.

It is guaranteed that a compacted acceleration structure doesn't consume more space than a non-compacted acceleration structure.

Pre-compaction, it is guaranteed that the size requirements reported by [GetRaytracingAccelerationStructurePrebuildInfo](#) for a given build configuration (triangle counts etc.) will be sufficient to store any acceleration structure whose build inputs are reduced (e.g. reducing triangle counts). This non-increasing property for smaller builds does not apply post-compaction, however. In other words, it is not guaranteed that having fewer items in an acceleration structure means it compresses to a smaller size than compressing an acceleration structure with more items.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC structure (d3d12.h)

Article 02/22/2024

Describes the space currently used by an acceleration structure..

Syntax

C++

```
typedef struct
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC {
    UINT64 CurrentSizeInBytes;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC;
```

Members

CurrentSizeInBytes

Space currently used by an acceleration structure. If the acceleration structure hasn't had a compaction operation performed on it, this size is the same one reported by [GetRaytracingAccelerationStructurePrebuildInfo](#), and if it has been compacted this size is the same reported for post-build info with [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE](#).

Remarks

The information in this structure is useful for tools to be able to determine how much memory is occupied by an arbitrary acceleration structure currently sitting in memory.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC structure (d3d12.h)

Article 02/22/2024

Description of the post-build information to generate from an acceleration structure. Use this structure in calls to [EmitRaytracingAccelerationStructurePostbuildInfo](#) and [BuildRaytracingAccelerationStructure](#).

Syntax

C++

```
typedef struct D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC {
    D3D12_GPU_VIRTUAL_ADDRESS DestBuffer;
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE InfoType;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC;
```

Members

DestBuffer

Storage for the post-build info result. Size required and the layout of the contents written by the system depend on the value of the *InfoType* field.

The memory pointed to must be in state

[D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#). The memory must be aligned to the natural alignment for the members of the particular output structure being generated (e.g. 8 bytes for a struct with the largest members being `UINT64`).

InfoType

A [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE](#) value specifying the type of post-build information to retrieve.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC structure (d3d12.h)

Article 02/22/2024

Describes the size and layout of the serialized acceleration structure and header

Syntax

C++

```
typedef struct
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC {
    UINT64 SerializedSizeInBytes;
    UINT64 NumBottomLevelAccelerationStructurePointers;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC;
```

Members

`SerializedSizeInBytes`

The size of the serialized acceleration structure, including a header. The header is [D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER](#) followed by followed by a list of pointers to bottom-level acceleration structures.

`NumBottomLevelAccelerationStructurePointers`

The number of 64-bit GPU virtual addresses that will be at the start of the serialized acceleration structure, after the

[D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER](#). For a bottom-level acceleration structure this will be 0. For a top-level acceleration structure, the pointers indicate the acceleration structures being referred to.

When deserialization occurs, these pointers to bottom-level pointers must be initialized by the app in the serialized data (just after the header) to the new locations where the bottom level acceleration structures will reside. It is not required that these new locations to have already been populated with bottom-level acceleration structures at deserialization time, as long as they are initialized with the expected deserialized data structures before being used in raytracing. During deserialization, the driver reads the

new pointers, using them to produce an equivalent top-level acceleration structure to the original.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC structure (d3d12.h)

Article02/22/2024

Describes the space requirement for decoding an acceleration structure into a form that can be visualized by tools.

Syntax

C++

```
typedef struct
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_D
ESC {
    UINT64 DecodedSizeInBytes;
}
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_D
ESC;
```

Members

DecodedSizeInBytes

The space requirement for decoding an acceleration structure into a form that can be visualized by tools.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of acceleration structure post-build info that can be retrieved with calls to [EmitRaytracingAccelerationStructurePostbuildInfo](#) and [BuildRaytracingAccelerationStructure](#).

Syntax

C++

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE = 0,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION
= 0x1,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION =
0x2,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE = 0x3
} ;
```

Constants

[+] Expand table

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE`

Value: 0

The post-build info is space requirements for an acceleration structure after compaction. For more information, see

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC](#).

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION`

Value: 0x1

The post-build info is space requirements for generating tools visualization for an acceleration structure. For more information, see

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC](#).

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION`

Value: 0x2

The post-build info is space requirements for serializing an acceleration structure. For more information, see

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC](#).

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE`

Value: `0x3`

The post-build info is size of the current acceleration structure. For more information, see

[D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO structure (d3d12.h)

Article 02/22/2024

Represents prebuild information about a raytracing acceleration structure. Get an instance of this structure by calling [GetRaytracingAccelerationStructurePrebuildInfo](#).

Syntax

C++

```
typedef struct D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO {
    UINT64 ResultDataMaxSizeInBytes;
    UINT64 ScratchDataSizeInBytes;
    UINT64 UpdateScratchDataSizeInBytes;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO;
```

Members

`ResultDataMaxSizeInBytes`

Size required to hold the result of an acceleration structure build based on the specified inputs.

`ScratchDataSizeInBytes`

Scratch storage on the GPU required during acceleration structure build based on the specified inputs.

UpdateScratchDataSizeInBytes

Scratch storage on GPU required during an acceleration structure update based on the specified inputs. This only needs to be called for the original acceleration structure build, and defines the scratch storage requirement for every acceleration structure update, other than the initial build.

If the [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE](#) flag is not specified when calling [GetRaytracingAccelerationStructurePrebuildInfo](#), the

returned value of this field is 0.

`UpdateScratchDataSizeInBytes`

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV structure (d3d12.h)

Article 02/22/2024

A shader resource view (SRV) structure for storing a raytracing acceleration structure.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV {
    D3D12_GPU_VIRTUAL_ADDRESS Location;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV;
```

Members

Location

The GPU virtual address of the SRV.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE enumeration (d3d12.h)

Article01/31/2022

Specifies the type of a raytracing acceleration structure.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL = 0,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL = 0x1
};
```

Constants

 Expand table

<code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL</code>

Value: 0

Top-level acceleration structure.

<code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL</code>
--

Value: 0x1

Bottom-level acceleration structure.

Remarks

Bottom-level acceleration structures each consist of a set of geometries that are building blocks for a scene. A top-level acceleration structure represents a set of instances of bottom-level acceleration structures.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_GEOMETRY_AABB_S_DESC structure (d3d12.h)

Article 02/22/2024

Describes a set of Axis-aligned bounding boxes that are used in the [D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#) structure to provide input data to a raytracing acceleration structure build operation.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_GEOMETRY_AABBS_DESC {
    UINT64                      AABBCount;
    D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE AABBs;
} D3D12_RAYTRACING_GEOMETRY_AABBS_DESC;
```

Members

AABBCount

The number of AABBs pointed to in the contiguous array at *AABBs*.

AABBs

the GPU memory location where an array of AABB descriptions is to be found, including the data stride between AABBs. The address and stride must each be aligned to 8 bytes, defined as The address must be aligned to 16 bytes, defined as [D3D12_RAYTRACING_AABB_BYTE_ALIGNMENT](#). The stride can be 0.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_GEOMETRY_DESC structure (d3d12.h)

Article 02/22/2024

Describes a set of geometry that is used in the [D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#) structure to provide input data to a raytracing acceleration structure build operation.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_GEOMETRY_DESC {
    D3D12_RAYTRACING_GEOMETRY_TYPE Type;
    D3D12_RAYTRACING_GEOMETRY_FLAGS Flags;
    union {
        D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC Triangles;
        D3D12_RAYTRACING_GEOMETRY_AABBS_DESC     AABBs;
    };
} D3D12_RAYTRACING_GEOMETRY_DESC;
```

Members

Type

The type of geometry.

Flags

The geometry flags

Triangles

A [D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC](#) describing triangle geometry, if *Type* is [D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES](#). Otherwise this parameter is unused.

AABBs

A [D3D12_RAYTRACING_GEOMETRY_AABBS_DESC](#) describing triangle geometry, if *Type* is [D3D12_RAYTRACING_GEOMETRY_TYPE PROCEDURAL_PRIMITIVE_AABBS](#). Otherwise this parameter is unused.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_RAYTRACING_GEOMETRY_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies flags for raytracing geometry in a [D3D12_RAYTRACING_GEOMETRY_DESC](#) structure.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_GEOMETRY_FLAGS {
    D3D12_RAYTRACING_GEOMETRY_FLAG_NONE = 0,
    D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE = 0x1,
    D3D12_RAYTRACING_GEOMETRY_FLAG_NO_DUPLICATE_ANYHIT_INVOCATION = 0x2
};
```

Constants

[+] [Expand table](#)

`D3D12_RAYTRACING_GEOMETRY_FLAG_NONE`

Value: *0*

No options specified.

`D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE`

Value: *0x1*

When rays encounter this geometry, the geometry acts as if no any hit shader is present. It is recommended that apps use this flag liberally, as it can enable important ray-processing optimizations. Note that this behavior can be overridden on a per-instance basis with [D3D12_RAYTRACING_INSTANCE_FLAGS](#) and on a per-ray basis using ray flags in [TraceRay](#).

`D3D12_RAYTRACING_GEOMETRY_FLAG_NO_DUPLICATE_ANYHIT_INVOCATION`

Value: *0x2*

By default, the system is free to trigger an any hit shader more than once for a given ray-primitive intersection. This flexibility helps improve the traversal efficiency of acceleration structures in certain cases. For instance, if the acceleration structure is implemented internally with bounding volumes, the implementation may find it beneficial to store relatively long triangles in multiple bounding boxes rather than a larger single box. However, some application use cases require that intersections be reported to the any hit shader at most once. This flag enables that guarantee for the given geometry, potentially with some performance impact.

This flag applies to all geometry types.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC structure (d3d12.h)

Article 07/23/2021

Describes a set of triangles used as raytracing geometry. The geometry pointed to by this struct are always in triangle list form, indexed or non-indexed. Triangle strips are not supported.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC {
    D3D12_GPU_VIRTUAL_ADDRESS           Transform3x4;
    DXGI_FORMAT                         IndexFormat;
    DXGI_FORMAT                         VertexFormat;
    UINT                                IndexCount;
    UINT                                VertexCount;
    D3D12_GPU_VIRTUAL_ADDRESS           IndexBuffer;
    D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE VertexBuffer;
} D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC;
```

Members

Transform3x4

Address of a 3x4 affine transform matrix in row-major layout to be applied to the vertices in the *VertexBuffer* during an acceleration structure build. The contents of *VertexBuffer* are not modified. If a 2D vertex format is used, the transformation is applied with the third vertex component assumed to be zero.

If *Transform3x4* is NULL the vertices will not be transformed. Using *Transform3x4* may result in increased computation and/or memory requirements for the acceleration structure build.

The memory pointed to must be in state

[D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#). The address must be aligned to 16 bytes, defined as

[D3D12_RAYTRACING_TRANSFORM3X4_BYTE_ALIGNMENT](#).

IndexFormat

Format of the indices in the *IndexBuffer*. Must be one of the following:

- **DXGI_FORMAT_UNKNOWN** - when *IndexBuffer* is NULL
- **DXGI_FORMAT_R32_UINT**
- **DXGI_FORMAT_R16_UINT**

VertexFormat

Format of the vertices in *VertexBuffer*. Must be one of the following:

- **DXGI_FORMAT_R32G32_FLOAT** - third component is assumed 0
- **DXGI_FORMAT_R32G32B32_FLOAT**
- **DXGI_FORMAT_R16G16_FLOAT** - third component is assumed 0
- **DXGI_FORMAT_R16G16B16A16_FLOAT** - A16 component is ignored, other data can be packed there, such as setting vertex stride to 6 bytes.
- **DXGI_FORMAT_R16G16_SNORM** - third component is assumed 0
- **DXGI_FORMAT_R16G16B16A16_SNORM** - A16 component is ignored, other data can be packed there, such as setting vertex stride to 6 bytes.

Tier 1.1 devices support the following additional formats:

- **DXGI_FORMAT_R16G16B16A16_UNORM** - A16 component is ignored, other data can be packed there, such as setting vertex stride to 6 bytes
- **DXGI_FORMAT_R16G16_UNORM** - third component assumed 0
- **DXGI_FORMAT_R10G10B10A2_UNORM** - A2 component is ignored, stride must be 4 bytes
- **DXGI_FORMAT_R8G8B8A8_UNORM** - A8 component is ignored, other data can be packed there, such as setting vertex stride to 3 bytes
- **DXGI_FORMAT_R8G8_UNORM** - third component assumed 0
- **DXGI_FORMAT_R8G8B8A8_SNORM** - A8 component is ignored, other data can be packed there, such as setting vertex stride to 3 bytes
- **DXGI_FORMAT_R8G8_SNORM** - third component assumed 0

IndexCount

Number of indices in *IndexBuffer*. Must be 0 if *IndexBuffer* is NULL.

VertexCount

Number of vertices in *VertexBuffer*.

IndexBuffer

Array of vertex indices. If NULL, triangles are non-indexed. Just as with graphics, the address must be aligned to the size of *IndexFormat*.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#). Note that if an app wants to share index buffer inputs between graphics input assembler and raytracing acceleration structure build input, it can always put a resource into a combination of read states simultaneously, e.g. [D3D12_RESOURCE_STATE_INDEX_BUFFER](#) | [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#).

VertexBuffer

Array of vertices including a stride. The alignment on the address and stride must be a multiple of the component size, so 4 bytes for formats with 32bit components and 2 bytes for formats with 16bit components. Unlike graphics, there is no constraint on the stride, other than that the bottom 32bits of the value are all that are used – the field is `UINT64` purely to make neighboring fields align cleanly/obviously everywhere. Each vertex position is expected to be at the start address of the stride range and any excess space is ignored by acceleration structure builds. This excess space might contain other app data such as vertex attributes, which the app is responsible for manually fetching in shaders, whether it is interleaved in vertex buffers or elsewhere.

The memory pointed to must be in state [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#). Note that if an app wants to share vertex buffer inputs between graphics input assembler and raytracing acceleration structure build input, it can always put a resource into a combination of read states simultaneously, e.g. [D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER](#) | [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#)

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_GEOMETRY_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of geometry used for raytracing. Use a value from this enumeration to specify the geometry type in a [D3D12_RAYTRACING_GEOMETRY_DESC](#).

Syntax

C++

```
typedef enum D3D12_RAYTRACING_GEOMETRY_TYPE {
    D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES = 0,
    D3D12_RAYTRACING_GEOMETRY_TYPE PROCEDURAL_PRIMITIVE_AABBS
};
```

Constants

[+] Expand table

`D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES`

Value: 0

The geometry consists of triangles.

`D3D12_RAYTRACING_GEOMETRY_TYPE PROCEDURAL_PRIMITIVE_AABBS`

The geometry procedurally is defined during raytracing by intersection shaders. For the purpose of acceleration structure builds, the geometry's bounds are described with axis-aligned bounding boxes using the [D3D12_RAYTRACING_GEOMETRY_AABBS_DESC](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_INSTANCE_DESC structure (d3d12.h)

Article 02/22/2024

Describes an instance of a raytracing acceleration structure used in GPU memory during the acceleration structure build process.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_INSTANCE_DESC {
    FLOAT             Transform[3][4];
    UINT              InstanceID : 24;
    UINT              InstanceMask : 8;
    UINT              InstanceContributionToHitGroupIndex : 24;
    UINT              Flags : 8;
    D3D12_GPU_VIRTUAL_ADDRESS AccelerationStructure;
} D3D12_RAYTRACING_INSTANCE_DESC;
```

Members

Transform[3]

Type: **FLOAT** [3][4]

A 3x4 transform matrix in row-major layout representing the instance-to-world transformation. Implementations transform rays, as opposed to transforming all of the geometry or AABBs.

ⓘ Note

The layout of `Transform` is a transpose of how affine matrices are typically stored in memory. Instead of four 3-vectors, `Transform` is laid out as three 4-vectors.

InstanceID

Type: **UINT** : 24

An arbitrary 24-bit value that can be accessed using the `InstanceID` intrinsic function in supported shader types.

`InstanceMask`

Type: `UINT : 8`

An 8-bit mask assigned to the instance, which can be used to include/reject groups of instances on a per-ray basis. If the value is zero, then the instance will never be included, so typically this should be set to some non-zero value. For more information see, the `InstanceInclusionMask` parameter to the `TraceRay` function.

`InstanceContributionToHitGroupIndex`

Type: `UINT : 24`

An arbitrary 24-bit value representing per-instance contribution to add into shader table indexing to select the hit group to use.

`Flags`

Type: `UINT : 8`

An 8-bit mask representing flags from `D3D12_RAYTRACING_INSTANCE_FLAGS` to apply to the instance.

`AccelerationStructure`

Type: `D3D12_GPU_VIRTUAL_ADDRESS`

Address of the bottom-level acceleration structure that is being instanced. The address must be aligned to 256 bytes, defined as

`D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT`. Any existing acceleration structure passed in here would already have been required to be placed with such alignment.

The memory pointed to must be in state

`D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE`.

Remarks

This C++ struct definition is useful if you're generating instance data on the CPU first, then uploading to the GPU. But your application is also free to generate instance descriptions directly into GPU memory (from compute shaders, for instance) following the same layout.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_RAYTRACING_INSTANCE_FLAGS enumeration (d3d12.h)

Article01/31/2022

Flags for a raytracing acceleration structure instance. These flags can be used to override [D3D12_RAYTRACING_GEOMETRY_FLAGS](#) for individual instances.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_INSTANCE_FLAGS {
    D3D12_RAYTRACING_INSTANCE_FLAG_NONE = 0,
    D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE = 0x1,
    D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_FRONT_COUNTERCLOCKWISE = 0x2,
    D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_OPAQUE = 0x4,
    D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE = 0x8
};
```

Constants

[] [Expand table](#)

`D3D12_RAYTRACING_INSTANCE_FLAG_NONE`

Value: *0*

No options specified.

`D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE`

Value: *0x1*

Disables front/back face culling for this instance. The Ray flags

`RAY_FLAG_CULL_BACK_FACING_TRIANGLES` and `RAY_FLAG_CULL_FRONT_FACING_TRIANGLES` will have no effect on this instance.

`D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_FRONT_COUNTERCLOCKWISE`

Value: *0x2*

This flag reverses front and back facings, which is useful if the application's natural winding order differs from the default. By default, a triangle is front facing if its vertices appear clockwise from the ray origin and back facing if its vertices appear counter-clockwise from the ray origin, in object space in a left-handed coordinate system.

Since these winding direction rules are defined in object space, they are unaffected by instance

transforms. For example, an instance transform matrix with negative determinant (e.g. mirroring some geometry) does not change the facing of the triangles within the instance. Per-geometry transforms defined in [D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC](#), by contrast, get combined with the associated vertex data in object space, so a negative determinant matrix there does flip triangle winding.

D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_OPAQUE

Value: 0x4

The instance will act as if [D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE](#) had been specified for all the geometries in the bottom-level acceleration structure referenced by the instance. Note that this behavior can be overridden by the ray flag [RAY_FLAG_FORCE_NON_OPAQUE](#).

This flag is mutually exclusive to the

[D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE](#) flag.

D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE

Value: 0x8

The instance will act as if [D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE](#) had not been specified for any of the geometries in the bottom-level acceleration structure referenced by the instance. Note that this behavior can be overridden by the ray flag [RAY_FLAG_FORCE_OPAQUE](#).

This flag is mutually exclusive to the [D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_OPAQUE](#) flag.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_PIPELINE_CONFIG structure (d3d12.h)

Article 02/22/2024

A state subobject that represents a raytracing pipeline configuration.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_PIPELINE_CONFIG {
    UINT MaxTraceRecursionDepth;
} D3D12_RAYTRACING_PIPELINE_CONFIG;
```

Members

MaxTraceRecursionDepth

Type: [UINT](#)

Limit on ray recursion for the raytracing pipeline. It must be in the range of 0 to 31. Below the maximum recursion depth, shader invocations such as closest hit or miss shaders can call [TraceRay](#) any number of times. At the maximum recursion depth, [TraceRay](#) calls result in the device going into removed state.

Remarks

A raytracing pipeline needs one raytracing pipeline configuration. If multiple pipeline configurations are present, then they must all match in content. But there's no benefit to such duplication. For example, defining it once per collection doesn't help drivers do early shader compilation before a raytracing pipeline is created. This is unlike [D3D12_RAYTRACING_SHADER_CONFIG](#), which *does* benefit from duplication per collection.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_PIPELINE_CONFIG1 structure (d3d12.h)

Article 02/22/2024

A state subobject that represents a raytracing pipeline configuration, with flags.

D3D12_RAYTRACING_PIPELINE_CONFIG1 requires Tier 1.1 raytracing support (see [D3D12_RAYTRACING_TIER](#)).

Syntax

C++

```
typedef struct D3D12_RAYTRACING_PIPELINE_CONFIG1 {
    UINT                         MaxTraceRecursionDepth;
    D3D12_RAYTRACING_PIPELINE_FLAGS Flags;
} D3D12_RAYTRACING_PIPELINE_CONFIG1;
```

Members

MaxTraceRecursionDepth

Type: [UINT](#)

Limit on ray recursion for the raytracing pipeline. It must be in the range of 0 to 31. Below the maximum recursion depth, shader invocations such as closest hit or miss shaders can call [TraceRay](#) any number of times. At the maximum recursion depth, [TraceRay](#) calls result in the device going into removed state.

Flags

Type: [D3D12_RAYTRACING_PIPELINE_FLAGS](#)

Configuration flags for the raytracing pipeline.

Remarks

A raytracing pipeline needs one raytracing pipeline configuration. If multiple pipeline configurations are present, then they must all match in content. But there's no benefit to such duplication. For example, defining it once per collection doesn't help drivers do

early shader compilation before a raytracing pipeline is created. This is unlike [D3D12_RAYTRACING_SHADER_CONFIG](#), which *does* benefit from duplication per collection.

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_PIPELINE_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify configuration flags for a raytracing pipeline.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_PIPELINE_FLAGS {
    D3D12_RAYTRACING_PIPELINE_FLAG_NONE = 0,
    D3D12_RAYTRACING_PIPELINE_FLAG_SKIP_TRIANGLES = 0x100,
    D3D12_RAYTRACING_PIPELINE_FLAG_SKIP_PROCEDURAL_PRIMITIVES = 0x200
};
```

Constants

[] Expand table

<code>D3D12_RAYTRACING_PIPELINE_FLAG_NONE</code>
--

Value: *0*

Specifies no option.

<code>D3D12_RAYTRACING_PIPELINE_FLAG_SKIP_TRIANGLES</code>
--

Value: *0x100*

Specifies that for any `TraceRay` call within this raytracing pipeline, the `RAY_FLAG_SKIP_TRIANGLES` ray flag should be added in. The resulting combination of ray flags must be valid. The presence of this flag in a raytracing pipeline config doesn't show up in a `RayFlags` call from a shader.

Implementations might be able to optimize pipelines knowing that a particular primitive type need not be considered.

<code>D3D12_RAYTRACING_PIPELINE_FLAG_SKIP_PROCEDURAL_PRIMITIVES</code>
--

Value: *0x200*

Specifies that for any `TraceRay` call within this raytracing pipeline, the `RAY_FLAG_SKIP_PROCEDURAL_PRIMITIVES` ray flag should be added in.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_SHADER_CONFIG structure (d3d12.h)

Article02/22/2024

A state subobject that represents a shader configuration.

Syntax

C++

```
typedef struct D3D12_RAYTRACING_SHADER_CONFIG {
    UINT MaxPayloadSizeInBytes;
    UINT MaxAttributeSizeInBytes;
} D3D12_RAYTRACING_SHADER_CONFIG;
```

Members

`MaxPayloadSizeInBytes`

The maximum storage for scalars (counted as 4 bytes each) in ray payloads in raytracing pipelines that contain this program.

`MaxAttributeSizeInBytes`

The maximum number of scalars (counted as 4 bytes each) that can be used for attributes in pipelines that contain this shader. The value cannot exceed [D3D12_RAYTRACING_MAX_ATTRIBUTE_SIZE_IN_BYTES](#).

Remarks

A raytracing pipeline needs one raytracing shader configuration. If multiple shader configurations are present, such as one in each collection to enable independent driver compilation for each one, they must all match when combined into a raytracing pipeline.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RAYTRACING_TIER enumeration (d3d12.h)

Article02/22/2024

Specifies the level of ray tracing support on the graphics device.

Syntax

C++

```
typedef enum D3D12_RAYTRACING_TIER {
    D3D12_RAYTRACING_TIER_NOT_SUPPORTED = 0,
    D3D12_RAYTRACING_TIER_1_0 = 10,
    D3D12_RAYTRACING_TIER_1_1
} ;
```

Constants

 Expand table

`D3D12_RAYTRACING_TIER_NOT_SUPPORTED`

Value: 0

No support for ray tracing on the device. Attempts to create any ray tracing-related object will fail, and using ray tracing-related APIs on command lists results in undefined behavior.

`D3D12_RAYTRACING_TIER_1_0`

Value: 10

The device supports tier 1 ray tracing functionality. In the current release, this tier represents all available ray tracing features.

Remarks

To determine the supported ray tracing tier for a graphics device, pass [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) struct.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_RENDER_PASS_BEGINNING_ACCESS structure (d3d12.h)

Article 02/22/2024

Describes the access to resource(s) that is requested by an application at the transition into a render pass.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_BEGINNING_ACCESS {
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE Type;
    union {
        D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS           Clear;
        D3D12_RENDER_PASS_BEGINNING_ACCESS_PRESERVE_LOCAL_PARAMETERS
    PreserveLocal;
    };
} D3D12_RENDER_PASS_BEGINNING_ACCESS;
```

Members

Type

A [D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE](#). The type of access being requested.

Clear

A [D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS](#). Appropriate when **Type** is [D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_CLEAR](#). The clear value to which resource(s) should be cleared.

PreserveLocal

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS structure (d3d12.h)

Article02/22/2024

Describes the clear value to which resource(s) should be cleared at the beginning of a render pass.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS {
    D3D12_CLEAR_VALUE ClearValue;
} D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS;
```

Members

`ClearValue`

A [D3D12_CLEAR_VALUE](#). The clear value to which the resource(s) should be cleared.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE enumeration (d3d12.h)

Article02/10/2023

Specifies the type of access that an application is given to the specified resource(s) at the transition into a render pass.

Syntax

C++

```
typedef enum D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE {
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_DISCARD = 0,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_CLEAR,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_NO_ACCESS,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE_LOCAL_RENDER,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE_LOCAL_SRV,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE_LOCAL_UAV
} ;
```

Constants

[] Expand table

`D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_DISCARD`

Value: 0

Indicates that your application doesn't have any dependency on the prior contents of the resource(s). You also shouldn't have any expectations about those contents, because a display driver may return the previously-written contents, or it may return uninitialized data. You can be assured that reading from the resource(s) won't hang the GPU, even if you do get undefined data back.

A read is defined as a traditional read from an unordered access view (UAV), a shader resource view (SRV), a constant buffer view (CBV), a vertex buffer view (VBV), an index buffer view (IBV), an IndirectArg binding/read, or a blend/depth-testing-induced read.

`D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE`

Indicates that your application has a dependency on the prior contents of the resource(s), so the contents must be loaded from main memory.

D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_CLEAR

Indicates that your application needs the resource(s) to be cleared to a specific value (a value that your application specifies). This clear occurs whether or not you interact with the resource(s) during the render pass. You specify the clear value at [BeginRenderPass](#) time, in the **Clear** member of your [D3D12_RENDER_PASS_BEGINNING_ACCESS](#) structure.

D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_NO_ACCESS

Indicates that your application will neither read from nor write to the resource(s) during the render pass. You would most likely use this value to indicate that you won't be accessing the depth/stencil plane for a depth/stencil view (DSV). You must pair this value with [D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_NO_ACCESS](#) in the corresponding [D3D12_RENDER_PASS_ENDING_ACCESS](#) structure.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_DEPTH_STENCIL_DESC structure (d3d12.h)

Article 02/22/2024

Describes a binding (fixed for the duration of the render pass) to a depth stencil view (DSV), as well as its beginning and ending access characteristics.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_DEPTH_STENCIL_DESC {
    D3D12_CPU_DESCRIPTOR_HANDLE      cpuDescriptor;
    D3D12_RENDER_PASS_BEGINNING_ACCESS DepthBeginningAccess;
    D3D12_RENDER_PASS_BEGINNING_ACCESS StencilBeginningAccess;
    D3D12_RENDER_PASS_ENDING_ACCESS   DepthEndingAccess;
    D3D12_RENDER_PASS_ENDING_ACCESS   StencilEndingAccess;
} D3D12_RENDER_PASS_DEPTH_STENCIL_DESC;
```

Members

`cpuDescriptor`

A [D3D12_CPU_DESCRIPTOR_HANDLE](#). The CPU descriptor handle corresponding to the depth stencil view (DSV).

`DepthBeginningAccess`

A [D3D12_RENDER_PASS_BEGINNING_ACCESS](#). The access to the depth buffer requested at the transition into a render pass.

`StencilBeginningAccess`

A [D3D12_RENDER_PASS_BEGINNING_ACCESS](#). The access to the stencil buffer requested at the transition into a render pass.

`DepthEndingAccess`

A [D3D12_RENDER_PASS_ENDING_ACCESS](#). The access to the depth buffer requested at the transition out of a render pass.

`StencilEndingAccess`

A [D3D12_RENDER_PASS_ENDING_ACCESS](#). The access to the stencil buffer requested at the transition out of a render pass.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

[\[+\] Yes](#)

[\[+\] No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_ENDING_ACCESS structure (d3d12.h)

Article02/22/2024

Describes the access to resource(s) that is requested by an application at the transition out of a render pass.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_ENDING_ACCESS {
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE Type;
    union {
        D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS Resolve;
        D3D12_RENDER_PASS_ENDING_ACCESS_PRESERVE_LOCAL_PARAMETERS PreserveLocal;
    };
} D3D12_RENDER_PASS_ENDING_ACCESS;
```

Members

Type

A [D3D12_RENDER_PASS_ENDING_ACCESS_TYPE](#). The type of access being requested.

Resolve

A [D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS](#). Appropriate when **Type** is [D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_RESOLVE](#). Description of the resource to resolve to.

PreserveLocal

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS structure (d3d12.h)

Article 02/22/2024

Describes a resource to resolve to at the conclusion of a render pass.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS {
    ID3D12Resource
    *pSrcResource;
    ID3D12Resource
    *pDstResource;
    UINT
    SubresourceCount;
    const D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS
    *pSubresourceParameters;
    DXGI_FORMAT
    Format;
    D3D12_RESOLVE_MODE
    ResolveMode;
    BOOL
    PreserveResolveSource;
} D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS;
```

Members

`pSrcResource`

A pointer to an [ID3D12Resource](#). The source resource.

`pDstResource`

A pointer to an [ID3D12Resource](#). The destination resource.

`SubresourceCount`

A `UINT`. The number of subresources.

`pSubresourceParameters`

A pointer to a constant array of [D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS](#). These subresources can be a subset of the render target's array slices, but you can't target subresources that aren't part of the render target view (RTV) or the depth/stencil view (DSV).

 **Note**

This pointer is directly referenced by the command list, and the memory for this array must remain alive and intact until [EndRenderPass](#) is called.

`Format`

A [DXGI_FORMAT](#). The data format of the resources.

`ResolveMode`

A [D3D12_RESOLVE_MODE](#). The resolve operation.

`PreserveResolveSource`

A `BOOL`. `TRUE` to preserve the resolve source, otherwise `FALSE`.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS structure (d3d12.h)

Article 02/22/2024

Describes the subresources involved in resolving at the conclusion of a render pass.

Syntax

C++

```
typedef struct
D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS {
    UINT      SrcSubresource;
    UINT      DstSubresource;
    UINT      DstX;
    UINT      DstY;
    D3D12_RECT SrcRect;
} D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS;
```

Members

SrcSubresource

A **UINT**. The source subresource.

DstSubresource

A **UINT**. The destination subresource.

DstX

A **UINT**. The x coordinate within the destination subresource.

DstY

A **UINT**. The y coordinate within the destination subresource.

SrcRect

A **D3D12_RECT**. The rectangle within the source subresource.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_ENDING_ACCESS_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of access that an application is given to the specified resource(s) at the transition out of a render pass.

Syntax

C++

```
typedef enum D3D12_RENDER_PASS_ENDING_ACCESS_TYPE {
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_DISCARD = 0,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_RESOLVE,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_NO_ACCESS,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE_LOCAL_RENDER,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE_LOCAL_SRV,
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE_LOCAL_UAV
} ;
```

Constants

[+] Expand table

`D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_DISCARD`

Value: 0

Indicates that your application won't have any future dependency on any data that you wrote to the resource(s) during this render pass. For example, a depth buffer that won't be textured from before it's written to again.

`D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE`

Indicates that your application will have a dependency on the written contents of the resource(s) in the future, and so they must be preserved.

`D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_RESOLVE`

Indicates that the resource(s)—for example, a multi-sample anti-aliasing (MSAA) surface—should be directly resolved to a separate resource at the conclusion of the render pass. For a tile-based deferred renderer (TBDR), this should ideally happen while the MSAA contents are still in the tile cache. You should ensure that the resolve destination is in the

[D3D12_RESOURCE_STATE_RESOLVE_DEST](#) resource state when the render pass ends. The resolve

source is left in its initial resource state at the time the render pass ends. A resolve operation submitted by a render pass doesn't implicitly change the state of any resource.

D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_NO_ACCESS

Indicates that your application will neither read from nor write to the resource(s) during the render pass. You would most likely use this value to indicate that you won't be accessing the depth/stencil plane for a depth/stencil view (DSV). You must pair this value with [D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_NO_ACCESS](#) in the corresponding [D3D12_RENDER_PASS_BEGINNING_ACCESS](#) structure.

Requirements

[\[\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies the nature of the render pass; for example, whether it is a suspending or a resuming render pass.

Syntax

C++

```
typedef enum D3D12_RENDER_PASS_FLAGS {
    D3D12_RENDER_PASS_FLAG_NONE = 0,
    D3D12_RENDER_PASS_FLAG_ALLOW_UAV_WRITES = 0x1,
    D3D12_RENDER_PASS_FLAG_SUSPENDING_PASS = 0x2,
    D3D12_RENDER_PASS_FLAG_RESUMING_PASS = 0x4,
    D3D12_RENDER_PASS_FLAG_BIND_READ_ONLY_DEPTH,
    D3D12_RENDER_PASS_FLAG_BIND_READ_ONLY_STENCIL
} ;
```

Constants

[+] Expand table

`D3D12_RENDER_PASS_FLAG_NONE`

Value: *0*

Indicates that the render pass has no special requirements.

`D3D12_RENDER_PASS_FLAG_ALLOW_UAV_WRITES`

Value: *0x1*

Indicates that writes to unordered access view(s) should be allowed during the render pass.

`D3D12_RENDER_PASS_FLAG_SUSPENDING_PASS`

Value: *0x2*

Indicates that this is a suspending render pass.

`D3D12_RENDER_PASS_FLAG_RESUMING_PASS`

Value: *0x4*

Indicates that this is a resuming render pass.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_RENDER_TARGET_DESC structure (d3d12.h)

Article 02/22/2024

Describes bindings (fixed for the duration of the render pass) to one or more render target views (RTVs), as well as their beginning and ending access characteristics.

Syntax

C++

```
typedef struct D3D12_RENDER_PASS_RENDER_TARGET_DESC {
    D3D12_CPU_DESCRIPTOR_HANDLE      cpuDescriptor;
    D3D12_RENDER_PASS_BEGINNING_ACCESS BeginningAccess;
    D3D12_RENDER_PASS_ENDING_ACCESS   EndingAccess;
} D3D12_RENDER_PASS_RENDER_TARGET_DESC;
```

Members

`cpuDescriptor`

A [D3D12_CPU_DESCRIPTOR_HANDLE](#). The CPU descriptor handle corresponding to the render target view(s) (RTVs).

`BeginningAccess`

A [D3D12_RENDER_PASS_BEGINNING_ACCESS](#). The access to the RTV(s) requested at the transition into a render pass.

`EndingAccess`

A [D3D12_RENDER_PASS_ENDING_ACCESS](#). The access to the RTV(s) requested at the transition out of a render pass.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_PASS_TIER enumeration (d3d12.h)

Article02/22/2024

Specifies the level of support for render passes on a graphics device.

Syntax

C++

```
typedef enum D3D12_RENDER_PASS_TIER {
    D3D12_RENDER_PASS_TIER_0 = 0,
    D3D12_RENDER_PASS_TIER_1 = 1,
    D3D12_RENDER_PASS_TIER_2 = 2
};
```

Constants

[\[+\] Expand table](#)

`D3D12_RENDER_PASS_TIER_0`

Value: 0

The user-mode display driver hasn't implemented render passes, and so the feature is provided only via software emulation. Render passes might not provide a performance advantage at this level of support.

`D3D12_RENDER_PASS_TIER_1`

Value: 1

The render passes feature is implemented by the user-mode display driver, and render target/depth buffer writes may be accelerated. Unordered access view (UAV) writes are not efficiently supported within the render pass.

`D3D12_RENDER_PASS_TIER_2`

Value: 2

The render passes feature is implemented by the user-mode display driver, render target/depth buffer writes may be accelerated, and unordered access view (UAV) writes (provided that writes in a render pass are not read until a subsequent render pass) are likely to be more efficient than issuing the same work without using a render pass.

Remarks

To determine the level of support for render passes for a graphics device, pass [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) struct.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_TARGET_BLEND_DESC structure (d3d12.h)

Article 07/27/2022

Describes the blend state for a render target.

Syntax

C++

```
typedef struct D3D12_RENDER_TARGET_BLEND_DESC {
    BOOL          BlendEnable;
    BOOL          LogicOpEnable;
    D3D12_BLEND   SrcBlend;
    D3D12_BLEND   DestBlend;
    D3D12_BLEND_OP BlendOp;
    D3D12_BLEND   SrcBlendAlpha;
    D3D12_BLEND   DestBlendAlpha;
    D3D12_BLEND_OP BlendOpAlpha;
    D3D12_LOGIC_OP LogicOp;
    UINT8         RenderTargetWriteMask;
} D3D12_RENDER_TARGET_BLEND_DESC;
```

Members

`BlendEnable`

Specifies whether to enable (or disable) blending. Set to **TRUE** to enable blending.

 **Note**

It's not valid for *LogicOpEnable* and *BlendEnable* to both be **TRUE**.

`LogicOpEnable`

Specifies whether to enable (or disable) a logical operation. Set to **TRUE** to enable a logical operation.

 **Note**

It's not valid for *LogicOpEnable* and *BlendEnable* to both be TRUE.

SrcBlend

A [D3D12_BLEND](#)-typed value that specifies the operation to perform on the RGB value that the pixel shader outputs. The **BlendOp** member defines how to combine the **SrcBlend** and **DestBlend** operations.

DestBlend

A [D3D12_BLEND](#)-typed value that specifies the operation to perform on the current RGB value in the render target. The **BlendOp** member defines how to combine the **SrcBlend** and **DestBlend** operations.

BlendOp

A [D3D12_BLEND_OP](#)-typed value that defines how to combine the **SrcBlend** and **DestBlend** operations.

SrcBlendAlpha

A [D3D12_BLEND](#)-typed value that specifies the operation to perform on the alpha value that the pixel shader outputs. Blend options that end in _COLOR are not allowed. The **BlendOpAlpha** member defines how to combine the **SrcBlendAlpha** and **DestBlendAlpha** operations.

DestBlendAlpha

A [D3D12_BLEND](#)-typed value that specifies the operation to perform on the current alpha value in the render target. Blend options that end in _COLOR are not allowed. The **BlendOpAlpha** member defines how to combine the **SrcBlendAlpha** and **DestBlendAlpha** operations.

BlendOpAlpha

A [D3D12_BLEND_OP](#)-typed value that defines how to combine the **SrcBlendAlpha** and **DestBlendAlpha** operations.

LogicOp

A [D3D12_LOGIC_OP](#)-typed value that specifies the logical operation to configure for the render target.

RenderTargetWriteMask

A combination of [D3D12_COLOR_WRITE_ENABLE](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies a write mask.

Remarks

ⓘ Note

It's not valid for *LogicOpEnable* and *BlendEnable* to both be **TRUE**.

You specify an array of [D3D12_RENDER_TARGET_BLEND_DESC](#) structures in the **RenderTarget** member of the [D3D12_BLEND_DESC](#) structure to describe the blend states for render targets; you can bind up to eight render targets to the [output-merger stage](#) at one time.

For info about how blending is done, see the [output-merger stage](#).

Here are the default values for blend state.

[+] Expand table

State	Default Value
BlendEnable	FALSE
LogicOpEnable	FALSE
SrcBlend	D3D12_BLEND_ONE
DestBlend	D3D12_BLEND_ZERO
BlendOp	D3D12_BLEND_OP_ADD
SrcBlendAlpha	D3D12_BLEND_ONE
DestBlendAlpha	D3D12_BLEND_ZERO
BlendOpAlpha	D3D12_BLEND_OP_ADD
LogicOp	D3D12_LOGIC_OP_NOOP
RenderTargetWriteMask	D3D12_COLOR_WRITE_ENABLE_ALL

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RENDER_TARGET_VIEW_DESC structure (d3d12.h)

Article04/02/2021

Describes the subresources from a resource that are accessible by using a render-target view.

Syntax

C++

```
typedef struct D3D12_RENDER_TARGET_VIEW_DESC {
    DXGI_FORMAT           Format;
    D3D12_RTV_DIMENSION ViewDimension;
    union {
        D3D12_BUFFER_RTV      Buffer;
        D3D12_TEX1D_RTV       Texture1D;
        D3D12_TEX1D_ARRAY_RTV Texture1DArray;
        D3D12_TEX2D_RTV       Texture2D;
        D3D12_TEX2D_ARRAY_RTV Texture2DArray;
        D3D12_TEX2DMS_RTV     Texture2DMS;
        D3D12_TEX2DMS_ARRAY_RTV Texture2DMSArray;
        D3D12_TEX3D_RTV       Texture3D;
    };
} D3D12_RENDER_TARGET_VIEW_DESC;
```

Members

Format

A [DXGI_FORMAT](#)-typed value that specifies the viewing format.

ViewDimension

A [D3D12_RTV_DIMENSION](#)-typed value that specifies how the render-target resource will be accessed. This type specifies how the resource will be accessed. This member also determines which _RTV to use in the following union.

Buffer

A [D3D12_BUFFER_RTV](#) structure that specifies which buffer elements can be accessed.

Texture1D

A [D3D12_TEX1D_RTV](#) structure that specifies the subresources in a 1D texture that can be accessed.

Texture1DArray

A [D3D12_TEX1D_ARRAY_RTV](#) structure that specifies the subresources in a 1D texture array that can be accessed.

Texture2D

A [D3D12_TEX2D_RTV](#) structure that specifies the subresources in a 2D texture that can be accessed.

Texture2DArray

A [D3D12_TEX2D_ARRAY_RTV](#) structure that specifies the subresources in a 2D texture array that can be accessed.

Texture2DMS

A [D3D12_TEX2DMS_RTV](#) structure that specifies a single subresource because a multisampled 2D texture only contains one subresource.

Texture2DMSArray

A [D3D12_TEX2DMS_ARRAY_RTV](#) structure that specifies the subresources in a multisampled 2D texture array that can be accessed.

Texture3D

A [D3D12_TEX3D_RTV](#) structure that specifies subresources in a 3D texture that can be accessed.

Remarks

Pass a render-target-view description into [ID3D12Device::CreateRenderTargetView](#) to create a render-target view.

A render-target view can't use the following formats:

- Any typeless format.
- DXGI_FORMAT_R32G32B32 if the view will be used to bind a buffer (vertex, index, constant, or stream-output).

If the format is set to DXGI_FORMAT_UNKNOWN, then the format of the resource that the view binds to the pipeline will be used.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESIDENCY_FLAGS enumeration (d3d12.h)

Article02/22/2024

Used with the EnqueueMakeResident function to choose how residency operations proceed when the memory budget is exceeded.

Syntax

C++

```
typedef enum D3D12_RESIDENCY_FLAGS {
    D3D12_RESIDENCY_FLAG_NONE = 0,
    D3D12_RESIDENCY_FLAG_DENY_OVERBUDGET = 0x1
} ;
```

Constants

[] Expand table

`D3D12_RESIDENCY_FLAG_NONE`

Value: `0`

Specifies the default residency policy, which allows residency operations to succeed regardless of the application's current memory budget. EnqueueMakeResident returns `E_OUTOFMEMORY` only when there is no memory available.

`D3D12_RESIDENCY_FLAG_DENY_OVERBUDGET`

Value: `0x1`

Specifies that the EnqueueMakeResident function should return `E_OUTOFMEMORY` when the residency operation would exceed the application's current memory budget.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESIDENCY_PRIORITY enumeration (d3d12.h)

Article02/22/2024

Specifies broad residency priority buckets useful for quickly establishing an application priority scheme.

Applications can assign priority values other than the five values present in this enumeration.

Syntax

C++

```
typedef enum D3D12_RESIDENCY_PRIORITY {
    D3D12_RESIDENCY_PRIORITY_MINIMUM = 0x28000000,
    D3D12_RESIDENCY_PRIORITY_LOW = 0x50000000,
    D3D12_RESIDENCY_PRIORITY_NORMAL = 0x78000000,
    D3D12_RESIDENCY_PRIORITY_HIGH = 0xa0010000,
    D3D12_RESIDENCY_PRIORITY_MAXIMUM = 0xc8000000
};
```

Constants

[] Expand table

	D3D12_RESIDENCY_PRIORITY_MINIMUM Value: <i>0x28000000</i> Indicates a minimum priority.
	D3D12_RESIDENCY_PRIORITY_LOW Value: <i>0x50000000</i> Indicates a low priority.
	D3D12_RESIDENCY_PRIORITY_NORMAL Value: <i>0x78000000</i> Indicates a normal, medium, priority.
	D3D12_RESIDENCY_PRIORITY_HIGH Value: <i>0xa0010000</i>

Indicates a high priority. Applications are discouraged from using priorities greater than this. For more information see [ID3D12Device1::SetResidencyPriority](#).

D3D12_RESIDENCY_PRIORITY_MAXIMUM

Value: 0xc8000000

Indicates a maximum priority. Applications are discouraged from using priorities greater than this; **D3D12_RESIDENCY_PRIORITY_MAXIMUM** is not guaranteed to be available. For more information see [ID3D12Device1::SetResidencyPriority](#)

Remarks

This enum is used by the [SetResidencyPriority](#) method.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOLVE_MODE enumeration (d3d12.h)

Article 02/22/2024

Specifies a resolve operation.

Syntax

C++

```
typedef enum D3D12_RESOLVE_MODE {
    D3D12_RESOLVE_MODE_DECOMPRESS = 0,
    D3D12_RESOLVE_MODE_MIN = 1,
    D3D12_RESOLVE_MODE_MAX = 2,
    D3D12_RESOLVE_MODE_AVERAGE = 3,
    D3D12_RESOLVE_MODE_ENCODE_SAMPLER_FEEDBACK,
    D3D12_RESOLVE_MODE_DECODE_SAMPLER_FEEDBACK
} ;
```

Constants

[\[+\] Expand table](#)

D3D12_RESOLVE_MODE_DECOMPRESS

Value: 0

Resolves compressed source samples to their uncompressed values. When using this operation, the source and destination resources must have the same sample count, unlike the min, max, and average operations that require the destination to have a sample count of 1.

D3D12_RESOLVE_MODE_MIN

Value: 1

Resolves the source samples to their minimum value. It can be used with any render target or depth stencil format.

D3D12_RESOLVE_MODE_MAX

Value: 2

Resolves the source samples to their maximum value. It can be used with any render target or depth stencil format.

D3D12_RESOLVE_MODE_AVERAGE

Value: 3

Resolves the source samples to their average value. It can be used with any non-integer render

target format, including the depth plane. It can't be used with integer render target formats, including the stencil plane.

Remarks

This enum is used by the [ID3D12GraphicsCommandList1::ResolveSubresourceRegion](#) function.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_ALIASING_BARRIER structure (d3d12.h)

Article02/22/2024

Describes the transition between usages of two different resources that have mappings into the same heap.

Syntax

C++

```
typedef struct D3D12_RESOURCE_ALIASING_BARRIER {
    ID3D12Resource *pResourceBefore;
    ID3D12Resource *pResourceAfter;
} D3D12_RESOURCE_ALIASING_BARRIER;
```

Members

`pResourceBefore`

A pointer to the [ID3D12Resource](#) object that represents the before resource used in the transition.

`pResourceAfter`

A pointer to the [ID3D12Resource](#) object that represents the after resource used in the transition.

Remarks

This structure is a member of the [D3D12_RESOURCE_BARRIER](#) structure.

Both the before and the after resources can be specified or one or both resources can be **NULL**, which indicates that any placed or reserved resource could cause aliasing.

Refer to the usage models described in [CreatePlacedResource](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_ALLOCATION_INFO structure (d3d12.h)

Article 02/22/2024

Describes parameters needed to allocate resources.

Syntax

C++

```
typedef struct D3D12_RESOURCE_ALLOCATION_INFO {
    UINT64 SizeInBytes;
    UINT64 Alignment;
} D3D12_RESOURCE_ALLOCATION_INFO;
```

Members

`SizeInBytes`

Type: [UINT64](#)

The size, in bytes, of the resource.

`Alignment`

Type: [UINT64](#)

The alignment value for the resource; one of 4KB (4096), 64KB (65536), or 4MB (4194304) alignment.

Remarks

This structure is used by the [ID3D12Device::GetResourceAllocationInfo](#) and [ID3D12Device::GetResourceAllocationInfo1](#) methods.

Requirements

[Expand table]

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RESOURCE_ALLOCATION_INFO](#)

[Core structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_ALLOCATION_INFO1 structure (d3d12.h)

Article 02/22/2024

Describes parameters needed to allocate resources, including offset.

Syntax

C++

```
typedef struct D3D12_RESOURCE_ALLOCATION_INFO1 {
    UINT64 Offset;
    UINT64 Alignment;
    UINT64 SizeInBytes;
} D3D12_RESOURCE_ALLOCATION_INFO1;
```

Members

Offset

Type: [UINT64](#)

The offset, in bytes, of the resource.

Alignment

Type: [UINT64](#)

The alignment value for the resource; one of 4KB (4096), 64KB (65536), or 4MB (4194304) alignment.

SizeInBytes

Type: [UINT64](#)

The size, in bytes, of the resource.

Remarks

This structure is used by the [ID3D12Device::GetResourceAllocationInfo1](#) method.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Core structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_BARRIER structure (d3d12.h)

Article 08/03/2021

Describes a resource barrier (transition in resource use).

Syntax

C++

```
typedef struct D3D12_RESOURCE_BARRIER {
    D3D12_RESOURCE_BARRIER_TYPE  Type;
    D3D12_RESOURCE_BARRIER_FLAGS Flags;
    union {
        D3D12_RESOURCE_TRANSITION_BARRIER Transition;
        D3D12_RESOURCE_ALIASING_BARRIER   Aliasing;
        D3D12_RESOURCE_UAV_BARRIER       UAV;
    };
} D3D12_RESOURCE_BARRIER;
```

Members

Type

A [D3D12_RESOURCE_BARRIER_TYPE](#)-typed value that specifies the type of resource barrier. This member determines which type to use in the union below.

Flags

Specifies a [D3D12_RESOURCE_BARRIER_FLAGS](#) enumeration constant such as for "begin only" or "end only".

Transition

A [D3D12_RESOURCE_TRANSITION_BARRIER](#) structure that describes the transition of subresources between different usages.

Members specify the before and after usages of the subresources.

Aliasing

A [D3D12_RESOURCE_ALIASING_BARRIER](#) structure that describes the transition between usages of two different resources that have mappings into the same heap.

A [D3D12_RESOURCE_UAV_BARRIER](#) structure that describes a resource in which all UAV accesses (reads or writes) must complete before any future UAV accesses (read or write) can begin.

Remarks

This structure is used by the [ID3D12GraphicsCommandList::ResourceBarrier](#) method.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_BARRIER_FLAGS enumeration (d3d12.h)

Article02/22/2024

Flags for setting split resource barriers.

Syntax

C++

```
typedef enum D3D12_RESOURCE_BARRIER_FLAGS {
    D3D12_RESOURCE_BARRIER_FLAG_NONE = 0,
    D3D12_RESOURCE_BARRIER_FLAG_BEGIN_ONLY = 0x1,
    D3D12_RESOURCE_BARRIER_FLAG_END_ONLY = 0x2
};
```

Constants

[+] Expand table

<code>D3D12_RESOURCE_BARRIER_FLAG_NONE</code>

Value: *0*

No flags.

<code>D3D12_RESOURCE_BARRIER_FLAG_BEGIN_ONLY</code>

Value: *0x1*

This starts a barrier transition in a new state, putting a resource in a temporary no-access condition.

<code>D3D12_RESOURCE_BARRIER_FLAG_END_ONLY</code>

Value: *0x2*

This barrier completes a transition, setting a new state and restoring active access to a resource.

Remarks

Split barriers allow a single transition to be split into begin and end halves (refer to [Multi-engine synchronization](#)).

This enum is used by the *Flags* member of the [D3D12_RESOURCE_BARRIER](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[ResourceBarrier](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_BARRIER_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies a type of resource barrier (transition in resource use) description.

Syntax

C++

```
typedef enum D3D12_RESOURCE_BARRIER_TYPE {
    D3D12_RESOURCE_BARRIER_TYPE_TRANSITION = 0,
    D3D12_RESOURCE_BARRIER_TYPE_ALIASING,
    D3D12_RESOURCE_BARRIER_TYPE_UAV
};
```

Constants

[+] Expand table

`D3D12_RESOURCE_BARRIER_TYPE_TRANSITION`

Value: 0

A transition barrier that indicates a transition of a set of subresources between different usages.
The caller must specify the before and after usages of the subresources.

`D3D12_RESOURCE_BARRIER_TYPE_ALIASING`

An aliasing barrier that indicates a transition between usages of 2 different resources that have mappings into the same tile pool. The caller can specify both the before and the after resource.
Note that one or both resources can be `NULL`, which indicates that any tiled resource could cause aliasing.

`D3D12_RESOURCE_BARRIER_TYPE_UAV`

An unordered access view (UAV) barrier that indicates all UAV accesses (reads or writes) to a particular resource must complete before any future UAV accesses (read or write) can begin.

Remarks

This enum is used in the `D3D12_RESOURCE_BARRIER_TYPE` structure. Use these values with the [ID3D12GraphicsCommandList::ResourceBarrier](#) method.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_BINDING_TIER enumeration (d3d12.h)

Article02/22/2024

Identifies the tier of resource binding being used.

Syntax

C++

```
typedef enum D3D12_RESOURCE_BINDING_TIER {
    D3D12_RESOURCE_BINDING_TIER_1 = 1,
    D3D12_RESOURCE_BINDING_TIER_2 = 2,
    D3D12_RESOURCE_BINDING_TIER_3 = 3
};
```

Constants

 Expand table

`D3D12_RESOURCE_BINDING_TIER_1`

Value: 1

Tier 1.

See [Hardware Tiers](#).

`D3D12_RESOURCE_BINDING_TIER_2`

Value: 2

Tier 2.

See [Hardware Tiers](#).

`D3D12_RESOURCE_BINDING_TIER_3`

Value: 3

Tier 3.

See [Hardware Tiers](#).

Remarks

This enum is used by the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Hardware Tiers](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_DESC structure (d3d12.h)

Article04/02/2021

Describes a resource, such as a texture. This structure is used extensively.

Syntax

C++

```
typedef struct D3D12_RESOURCE_DESC {
    D3D12_RESOURCE_DIMENSION Dimension;
    UINT64 Alignment;
    UINT64 Width;
    UINT Height;
    UINT16 DepthOrArraySize;
    UINT16 MipLevels;
    DXGI_FORMAT Format;
    DXGI_SAMPLE_DESC SampleDesc;
    D3D12_TEXTURE_LAYOUT Layout;
    D3D12_RESOURCE_FLAGS Flags;
} D3D12_RESOURCE_DESC;
```

Members

Dimension

One member of [D3D12_RESOURCE_DIMENSION](#), specifying the dimensions of the resource (for example, D3D12_RESOURCE_DIMENSION_TEXTURE1D), or whether it is a buffer ((D3D12_RESOURCE_DIMENSION_BUFFER)).

Alignment

Specifies the alignment.

Width

Specifies the width of the resource.

Height

Specifies the height of the resource.

DepthOrArraySize

Specifies the depth of the resource, if it is 3D, or the array size if it is an array of 1D or 2D resources.

MipLevels

Specifies the number of MIP levels.

Format

Specifies one member of [DXGI_FORMAT](#).

SampleDesc

Specifies a [DXGI_SAMPLE_DESC](#) structure.

Layout

Specifies one member of [D3D12_TEXTURE_LAYOUT](#).

Flags

Bitwise-OR'd flags, as [D3D12_RESOURCE_FLAGS](#) enumeration constants.

Remarks

Use this structure with:

- [ID3D12Resource::GetDesc](#)
- [ID3D12Device::GetResourceAllocationInfo](#)
- [ID3D12Device::CreateCommittedResource](#)
- [ID3D12Device::CreatePlacedResource](#)
- [ID3D12Device::CreateReservedResource](#)
- [ID3D12Device::GetCopyableFootprints](#)
- A number of the helper functions, refer to [Helper Structures and Functions for D3D12](#).

Two common resources are buffers and textures, which both use this structure, but with quite different uses of the fields.

Buffers

Buffers are a contiguous memory region. *Width* may be between 1 and either the *MaxGPUVirtualAddressBitsPerResource* field of

[D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT](#) for reserved resources or the *MaxGPUVirtualAddressBitsPerProcess* field for committed resources. However, exhaustion of GPU virtual address space, memory residency budget (see [IDXGIAdapter3::QueryVideoMemoryInfo](#)), and/or system memory may easily occur first.

Alignment must be 64KB (D3D12_DEFAULT_RESOURCE_PLACEMENT_ALIGNMENT) or 0, which is effectively 64KB.

Height, *DepthOrArraySize*, and *MipLevels* must be 1.

Format must be DXGI_FORMAT_UNKNOWN.

SampleDesc.Count must be 1 and *Quality* must be 0.

Layout must be D3D12_TEXTURE_LAYOUT_ROW_MAJOR, as buffer memory layouts are understood by applications and row-major texture data is commonly marshaled through buffers.

Flags must still be accurately filled out by applications for buffers, with minor exceptions. However, applications can use the most amount of capability support without concern about the efficiency impact on buffers. The flags field is meant to control properties related to textures.

Textures

Textures are a multi-dimensional arrangement of texels in a contiguous region of memory, heavily optimized to maximize bandwidth for rendering and sampling. Texture sizes are hard to predict and vary from adapter to adapter. Applications must use [ID3D12Device::GetResourceAllocationInfo](#) to accurately understand their size.

TEXTURE1D, TEXTURE2D, and TEXTURE3D are not supported orthogonally on every format. See the use of D3D12_FORMAT_SUPPORT1_TEXTURE1D, D3D12_FORMAT_SUPPORT1_TEXTURE2D, and D3D12_FORMAT_SUPPORT1_TEXTURE3D in [D3D12_FORMAT_SUPPORT1](#).

Width, *Height*, and *DepthOrArraySize* must be between 1 and the maximum dimension supported for the particular feature level and texture dimension. However, exhaustion of GPU virtual address space, memory residency budget (see [IDXGIAdapter3::QueryVideoMemoryInfo](#)), and/or system memory may easily occur first. For compressed formats, these dimensions are logical. For example:

- For TEXTURE1D:
 - *Width* must be less than or equal to D3D10_REQ_TEXTURE1D_U_DIMENSION on feature levels less than 11_0 and D3D11_REQ_TEXTURE1D_U_DIMENSION on

feature level 11_0 or greater.

- *Height* must be 1.
- *DepthOrArraySize* is interpreted as array size and must be less than or equal to D3D10_REQ_TEXTURE1D_ARRAY_AXIS_DIMENSION on feature levels less than 11_0 and D3D11_REQ_TEXTURE1D_ARRAY_AXIS_DIMENSION on feature levels 11_0 or greater.
- For TEXTURE2D:
 - *Width* and *Height* must be less than or equal to D3D10_REQ_TEXTURE2D_U_OR_V_DIMENSION on feature levels less than 11_0 and D3D11_REQ_TEXTURE2D_U_OR_V_DIMENSION or feature level 11_0 or greater.
 - *DepthOrArraySize* is interpreted as array size and must be less than or equal to D3D10_REQ_TEXTURE2D_ARRAY_AXIS_DIMENSION on feature levels less than 11_0 and D3D11_REQ_TEXTURE2D_ARRAY_AXIS_DIMENSION on feature levels 11_0 or greater.
- For TEXTURE3D:
 - *Width* and *Height* and *DepthOrArraySize* must be less than or equal to D3D10_REQ_TEXTURE3D_U_V_OR_W_DIMENSION on feature levels less than 11_0 and D3D11_REQ_TEXTURE3D_U_V_OR_W_DIMENSION on feature level 11_0 or greater.
 - *DepthOrArraySize* is interpreted as depth.

The following notes are for all texture sizes.

Alignment

Alignment may be one of 0, 4KB, 64KB or 4MB.

If *Alignment* is set to 0, the runtime will use 4MB for MSAA textures and 64KB for everything else. The application may choose smaller alignments than these defaults for a couple of texture types when the texture is small. Textures with UNKNOWN layout and MSAA may be created with 64KB alignment (if they pass the small size restriction detailed below).

Textures with UNKNOWN layout without MSAA and without render-target nor depth-stencil flags may be created with 4KB Alignment (again, passing the small size restriction).

Applications can create smaller aligned resources when the estimated size of the most-detailed mip level is a total of the larger alignment restriction or less. The runtime will use an architecture-independent mechanism of size-estimation, that mimics the way standard swizzle and D3D12 tiled resources are sized. However, the tile sizes will be of

the smaller alignment restriction for such calculations. Using the non-render-target and non-depth-stencil texture as an example, the runtime will assume near-equilateral tile shapes of 4KB, and calculate the number of tiles needed for the most-detailed mip level. If the number of tiles is equal to or less than 16, then the application can create a 4KB aligned resource. So, a mipped tex2d array of any array size and any number of mip levels can be 4KB, as long as the width and height are small enough for the particular format and MSAA.

MipLevels

MipLevels may be 0, or 1 to the maximum mip levels supported by the *Width*, *Height*, and *DepthOrArraySize* dimensions. When 0 is used, the API will automatically calculate the maximum mip levels supported and use that. But, some resource and heap properties preclude mip levels, so the app must specify the value as 1.

Refer to the D3D12_FORMAT_SUPPORT1_MIP field of [D3D12_FORMAT_SUPPORT1](#) for per-format restrictions. MSAA resources, textures with D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER, and heaps with D3D12_HEAP_FLAG_ALLOW_DISPLAY all preclude mip levels.

Format

Format must be a valid format supported at the feature level of the device.

SampleDesc

A *SampleDesc.Count* greater than 1 and/ or non-zero *Quality* are only supported for TEXTURE2D and when either D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET or D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL are set.

The following are unsupported:

- D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE,
- D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS,
- D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS,
- D3D12_HEAP_FLAG_ALLOW_DISPLAY

See [D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS](#) for determining valid *Count* and *Quality* values.

Requirements

Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Structures](#)

[D3D12_HEAP_FLAGS](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_DESC1 structure (d3d12.h)

Article 02/22/2024

Describes a resource, such as a texture, including a mip region. This structure is used in several methods.

Syntax

C++

```
typedef struct D3D12_RESOURCE_DESC1 {
    D3D12_RESOURCE_DIMENSION Dimension;
    UINT64                  Alignment;
    UINT64                  Width;
    UINT                     Height;
    UINT16                  DepthOrArraySize;
    UINT16                  MipLevels;
    DXGI_FORMAT              Format;
    DXGI_SAMPLE_DESC          SampleDesc;
    D3D12_TEXTURE_LAYOUT      Layout;
    D3D12_RESOURCE_FLAGS      Flags;
    D3D12_MIP_REGION          SamplerFeedbackMipRegion;
} D3D12_RESOURCE_DESC1;
```

Members

Dimension

One member of [D3D12_RESOURCE_DIMENSION](#), specifying the dimensions of the resource (for example, D3D12_RESOURCE_DIMENSION_TEXTURE1D), or whether it is a buffer ((D3D12_RESOURCE_DIMENSION_BUFFER)).

Alignment

Specifies the alignment.

Width

Specifies the width of the resource.

Height

Specifies the height of the resource.

DepthOrArraySize

Specifies the depth of the resource, if it is 3D, or the array size if it is an array of 1D or 2D resources.

MipLevels

Specifies the number of MIP levels.

Format

Specifies one member of [DXGI_FORMAT](#).

SampleDesc

Specifies a [DXGI_SAMPLE_DESC](#) structure.

Layout

Specifies one member of [D3D12_TEXTURE_LAYOUT](#).

Flags

Bitwise-OR'd flags, as [D3D12_RESOURCE_FLAGS](#) enumeration constants.

SamplerFeedbackMipRegion

A [D3D12_MIP_REGION](#) struct.

Remarks

For remarks, see [D3D12_RESOURCE_DESC](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
 - [Sampler feedback specification ↗](#)
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_DIMENSION enumeration (d3d12.h)

Article 02/22/2024

Identifies the type of resource being used.

Syntax

C++

```
typedef enum D3D12_RESOURCE_DIMENSION {
    D3D12_RESOURCE_DIMENSION_UNKNOWN = 0,
    D3D12_RESOURCE_DIMENSION_BUFFER = 1,
    D3D12_RESOURCE_DIMENSION_TEXTURE1D = 2,
    D3D12_RESOURCE_DIMENSION_TEXTURE2D = 3,
    D3D12_RESOURCE_DIMENSION_TEXTURE3D = 4
};
```

Constants

[+] Expand table

D3D12_RESOURCE_DIMENSION_UNKNOWN

Value: 0

Resource is of unknown type.

D3D12_RESOURCE_DIMENSION_BUFFER

Value: 1

Resource is a buffer.

D3D12_RESOURCE_DIMENSION_TEXTURE1D

Value: 2

Resource is a 1D texture.

D3D12_RESOURCE_DIMENSION_TEXTURE2D

Value: 3

Resource is a 2D texture.

D3D12_RESOURCE_DIMENSION_TEXTURE3D

Value: 4

Resource is a 3D texture.

Remarks

This enum is used by the [D3D12_RESOURCE_DESC](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_FLAGS enumeration (d3d12.h)

Article11/03/2022

Defines constants that specify options for working with resources.

Syntax

C++

```
typedef enum D3D12_RESOURCE_FLAGS {
    D3D12_RESOURCE_FLAG_NONE = 0,
    D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET = 0x1,
    D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL = 0x2,
    D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS = 0x4,
    D3D12_RESOURCE_FLAG_DENY_SHADER_RESOURCE = 0x8,
    D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER = 0x10,
    D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS = 0x20,
    D3D12_RESOURCE_FLAG_VIDEO_DECODE_REFERENCE_ONLY = 0x40,
    D3D12_RESOURCE_FLAG_VIDEO_ENCODE_REFERENCE_ONLY = 0x80,
    D3D12_RESOURCE_FLAG_RAYTRACING_ACCELERATION_STRUCTURE = 0x100
} ;
```

Constants

[+] Expand table

D3D12_RESOURCE_FLAG_NONE

Value: 0

No options are specified.

D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET

Value: 0x1

Allows a render target view to be created for the resource; and also enables the resource to transition into the state of [D3D12_RESOURCE_STATE_RENDER_TARGET](#). Some adapter architectures allocate extra memory for textures with this flag to reduce the effective bandwidth during common rendering. This characteristic may not be beneficial for textures that are never rendered to, nor is it available for textures compressed with BC formats. Your application should avoid setting this flag when rendering will never occur.

The following restrictions and interactions apply:

- Either the texture format must support render target capabilities at the current feature level. Or, when the format is a typeless format, a format within the same typeless group must support render target capabilities at the current feature level.
- Can't be set in conjunction with textures that have `D3D12_TEXTURE_LAYOUT_ROW_MAJOR` when `D3D12_FEATURE_DATA_D3D12_OPTIONS::CrossAdapterRowMajorTextureSupported` is `FALSE`, nor in conjunction with textures that have `D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE` when `D3D12_FEATURE_DATA_D3D12_OPTIONS::StandardSwizzle64KBSupported` is `FALSE`.
- Can't be used with 4KB alignment, `D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL`, nor usage with heaps that have `D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES`.

`D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL`

Value: `0x2`

Allows a depth stencil view to be created for the resource, as well as enables the resource to transition into the state of `D3D12_RESOURCE_STATE_DEPTH_WRITE` and/or `D3D12_RESOURCE_STATE_DEPTH_READ`. Most adapter architectures allocate extra memory for textures with this flag to reduce the effective bandwidth, and maximize optimizations for early depth-test. Your application should avoid setting this flag when depth operations will never occur.

The following restrictions and interactions apply:

- Either the texture format must support depth stencil capabilities at the current feature level. Or, when the format is a typeless format, a format within the same typeless group must support depth stencil capabilities at the current feature level.
- Can't be used with `D3D12_RESOURCE_DIMENSION_BUFFER`, 4KB alignment, `D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET`, `D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS`, `D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS`, `D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE`, `D3D12_TEXTURE_LAYOUT_ROW_MAJOR`, nor used with heaps that have `D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES` or `D3D12_HEAP_FLAG_ALLOW_DISPLAY`.
- Precludes usage of `WriteToSubresource` and `ReadFromSubresource`.
- Precludes GPU copying of a subregion. `CopyTextureRegion` must copy a whole subresource to or from resources with this flag.

`D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS`

Value: `0x4`

Allows an unordered access view to be created for the resource, as well as enables the resource to transition into the state of [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#). Some adapter architectures must resort to less efficient texture layouts in order to provide this functionality. If a texture is rarely used for unordered access, then it might be worth having two textures around and copying between them. One texture would have this flag, while the other wouldn't. Your application should avoid setting this flag when unordered access operations will never occur.

The following restrictions and interactions apply:

- Either the texture format must support unordered access capabilities at the current feature level. Or, when the format is a typeless format, a format within the same typeless group must support unordered access capabilities at the current feature level.
- Can't be set in conjunction with textures that have [D3D12_TEXTURE_LAYOUT_ROW_MAJOR](#) when [D3D12_FEATURE_DATA_D3D12_OPTIONS::CrossAdapterRowMajorTextureSupported](#) is `FALSE`, nor in conjunction with textures that have [D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE](#) when [D3D12_FEATURE_DATA_D3D12_OPTIONS::StandardSwizzle64KBSupported](#) is `FALSE`, nor when the feature level is less than 11.0.
- Can't be used with MSAA textures.

`D3D12_RESOURCE_FLAG_DENY_SHADER_RESOURCE`

Value: `0x8`

Disallows a shader resource view from being created for the resource, as well as disables the resource from transitioning into the state of

[D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#) or [D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE](#). Some adapter architectures gain bandwidth capacity for depth stencil textures when shader resource views are precluded. If a texture is rarely used for shader resources, then it might be worth having two textures around and copying between them. One texture would have this flag, while the other wouldn't. Your application should set this flag when depth stencil textures will never be used from shader resource views.

The following restrictions and interactions apply:

- Must be used with [D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#).

`D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER`

Value: `0x10`

Allows the resource to be used for cross-adapter data, as well as those features enabled by [D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS](#). Cross-

adapter resources commonly preclude techniques that reduce effective texture bandwidth during usage, and some adapter architectures might require different caching behavior. Your application should avoid setting this flag when the resource data will never be used with another adapter.

The following restrictions and interactions apply:

- Must be used with heaps that have [D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER](#).
- Can't be used with heaps that have [D3D12_HEAP_FLAG_ALLOW_DISPLAY](#).

`D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS`

Value: *0x20*

Allows a resource to be simultaneously accessed by multiple different queues, devices, or processes (for example, allows a resource to be used with [ResourceBarrier](#) transitions performed in more than one command list executing at the same time).

Simultaneous access allows multiple readers and one writer, as long as the writer doesn't concurrently modify the texels that other readers are accessing. Some adapter architectures can't leverage techniques to reduce effective texture bandwidth during usage.

However, your application should avoid setting this flag when multiple readers are not required during frequent, non-overlapping writes to textures. Use of this flag can compromise resource fences to perform waits, and prevent any compression being used with a resource.

The following restrictions and interactions apply:

- Can't be used with [D3D12_RESOURCE_DIMENSION_BUFFER](#); but buffers always have the properties represented by this flag.
- Can't be used with MSAA textures.
- Can't be used with [D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#).

`D3D12_RESOURCE_FLAG_VIDEO_DECODE_REFERENCE_ONLY`

Value: *0x40*

Specifies that this resource may be used only as a decode reference frame. It may be written to or read only by the video decode operation.

[D3D12_VIDEO_DECODE_TIER_1](#) and [D3D12_VIDEO_DECODE_TIER_2](#) may report [D3D12_VIDEO_DECODE_CONFIGURATION_FLAG_REFERENCE_ONLY_ALLOCATIONS_REQUIRED](#) in the [D3D12_FEATURE_DATA_VIDEO_DECODE_SUPPORT](#) structure configuration flag. If that happens, then your application must allocate reference frames with the

`D3D12_RESOURCE_FLAGS::D3D12_RESOURCE_FLAG_VIDEO_DECODE_REFERENCE_ONLY`
resource flag.

`D3D12_VIDEO_DECODE_TIER_3` must not set the
[`D3D12_VIDEO_DECODE_CONFIGURATION_FLAG_REFERENCE_ONLY_ALLOCATIONS_REQUIRED`
(`./d3d12video/ne-d3d12video-d3d12_video_decode_configuration_flags`) configuration flag, and
must not require the use of this resource flag.

`D3D12_RESOURCE_FLAG_VIDEO_ENCODE_REFERENCE_ONLY`

Value: `0x80`

Specifies that this resource may be used only as an encode reference frame. It may be written to or
read only by the video encode operation.

`D3D12_RESOURCE_FLAG_RAYTRACING_ACCELERATION_STRUCTURE`

Value: `0x100`

Requires the DirectX 12 Agility SDK 1.608.0 or later. Indicates that a buffer is to be used as a
raytracing acceleration structure. When using D3D12 Enhanced Barriers, this flag serves as a
replacement for `D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE`, since buffers no
longer have layouts/states.

Remarks

This enum is used by the `Flags` member of the [D3D12_RESOURCE_DESC](#).

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

- [Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

D3D12_RESOURCE_HEAP_TIER enumeration (d3d12.h)

Article02/22/2024

Specifies which resource heap tier the hardware and driver support.

Syntax

C++

```
typedef enum D3D12_RESOURCE_HEAP_TIER {
    D3D12_RESOURCE_HEAP_TIER_1 = 1,
    D3D12_RESOURCE_HEAP_TIER_2 = 2
};
```

Constants

[] Expand table

<code>D3D12_RESOURCE_HEAP_TIER_1</code>

Value: 1

Indicates that heaps can only support resources from a single resource category.

For the list of resource categories, see Remarks.

In tier 1, these resource categories are mutually exclusive and cannot be used with the same heap.

The resource category must be declared when creating a heap, using the correct

[D3D12_HEAP_FLAGS](#) enumeration constant.

Applications cannot create heaps with flags that allow all three categories.

<code>D3D12_RESOURCE_HEAP_TIER_2</code>

Value: 2

Indicates that heaps can support resources from all three categories.

For the list of resource categories, see Remarks.

In tier 2, these resource categories can be mixed within the same heap.

Applications may create heaps with flags that allow all three categories; but are not required to do so.

Applications may be written to support tier 1 and seamlessly run on tier 2.

Remarks

This enum is used by the `ResourceHeapTier` member of the `D3D12_FEATURE_DATA_D3D12_OPTIONS` structure.

This enum specifies which resource heap tier the hardware and driver support. Lower tiers require more heap attribution than greater tiers.

Resources can be categorized into the following types:

- Buffers
- Non-render target & non-depth stencil textures
- Render target or depth stencil textures

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_STATES enumeration (d3d12.h)

Article03/17/2023

Defines constants that specify the state of a resource regarding how the resource is being used.

Syntax

C++

```
typedef enum D3D12_RESOURCE_STATES {
    D3D12_RESOURCE_STATE_COMMON = 0,
    D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER = 0x1,
    D3D12_RESOURCE_STATE_INDEX_BUFFER = 0x2,
    D3D12_RESOURCE_STATE_RENDER_TARGET = 0x4,
    D3D12_RESOURCE_STATE_UNORDERED_ACCESS = 0x8,
    D3D12_RESOURCE_STATE_DEPTH_WRITE = 0x10,
    D3D12_RESOURCE_STATE_DEPTH_READ = 0x20,
    D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE = 0x40,
    D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE = 0x80,
    D3D12_RESOURCE_STATE_STREAM_OUT = 0x100,
    D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT = 0x200,
    D3D12_RESOURCE_STATE_COPY_DEST = 0x400,
    D3D12_RESOURCE_STATE_COPY_SOURCE = 0x800,
    D3D12_RESOURCE_STATE_RESOLVE_DEST = 0x1000,
    D3D12_RESOURCE_STATE_RESOLVE_SOURCE = 0x2000,
    D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE = 0x400000,
    D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE = 0x1000000,
    D3D12_RESOURCE_STATE_RESERVED_INTERNAL_8000,
    D3D12_RESOURCE_STATE_RESERVED_INTERNAL_4000,
    D3D12_RESOURCE_STATE_RESERVED_INTERNAL_100000,
    D3D12_RESOURCE_STATE_RESERVED_INTERNAL_40000000,
    D3D12_RESOURCE_STATE_RESERVED_INTERNAL_80000000,
    D3D12_RESOURCE_STATE_GENERIC_READ,
    D3D12_RESOURCE_STATE_ALL_SHADER_RESOURCE,
    D3D12_RESOURCE_STATE_PRESENT = 0,
    D3D12_RESOURCE_STATE_PREDICATION = 0x200,
    D3D12_RESOURCE_STATE_VIDEO_DECODE_READ = 0x10000,
    D3D12_RESOURCE_STATE_VIDEO_DECODE_WRITE = 0x20000,
    D3D12_RESOURCE_STATE_VIDEO_PROCESS_READ = 0x40000,
    D3D12_RESOURCE_STATE_VIDEO_PROCESS_WRITE = 0x80000,
    D3D12_RESOURCE_STATE_VIDEO_ENCODE_READ = 0x200000,
    D3D12_RESOURCE_STATE_VIDEO_ENCODE_WRITE = 0x800000
};
```

Constants

 Expand table

D3D12_RESOURCE_STATE_COMMON

Value: *0*

Your application should transition to this state only for accessing a resource across different graphics engine types.

Specifically, a resource must be in the COMMON state before being used on a COPY queue (when previously used on DIRECT/COMPUTE), and before being used on DIRECT/COMPUTE (when previously used on COPY). This restriction doesn't exist when accessing data between DIRECT and COMPUTE queues.

The COMMON state can be used for all usages on a Copy queue using the implicit state transitions. For more info, in [Multi-engine synchronization](#), find "common".

Additionally, textures must be in the COMMON state for CPU access to be legal, assuming the texture was created in a CPU-visible heap in the first place.

D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER

Value: *0x1*

A subresource must be in this state when it is accessed by the GPU as a vertex buffer or constant buffer. This is a read-only state.

D3D12_RESOURCE_STATE_INDEX_BUFFER

Value: *0x2*

A subresource must be in this state when it is accessed by the 3D pipeline as an index buffer. This is a read-only state.

D3D12_RESOURCE_STATE_RENDER_TARGET

Value: *0x4*

The resource is used as a render target. A subresource must be in this state when it is rendered to, or when it is cleared with [ID3D12GraphicsCommandList::ClearRenderTargetView](#).

This is a write-only state. To read from a render target as a shader resource, the resource must be in either [D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE](#) or [D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE](#).

D3D12_RESOURCE_STATE_UNORDERED_ACCESS

Value: *0x8*

The resource is used for unordered access. A subresource must be in this state when it is accessed by the GPU via an unordered access view. A subresource must also be in this state when it is cleared with [ID3D12GraphicsCommandList::ClearUnorderedAccessViewInt](#) or [ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat](#). This is a read/write state.

D3D12_RESOURCE_STATE_DEPTH_WRITE

Value: 0x10

D3D12_RESOURCE_STATE_DEPTH_WRITE is a state that is mutually exclusive with other states. You should use it for [ID3D12GraphicsCommandList::ClearDepthStencilView](#) when the flags (see [D3D12_CLEAR_FLAGS](#)) indicate a given subresource should be cleared (otherwise the subresource state doesn't matter), or when using it in a writable depth stencil view (see [D3D12_DSV_FLAGS](#)) when the PSO has depth write enabled (see [D3D12_DEPTH_STENCIL_DESC](#)).

D3D12_RESOURCE_STATE_DEPTH_READ

Value: 0x20

DEPTH_READ is a state that can be combined with other states. It should be used when the subresource is in a read-only depth stencil view, or when depth write of [D3D12_DEPTH_STENCIL_DESC](#) is disabled. It can be combined with other read states (for example, [D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE](#)), such that the resource can be used for the depth or stencil test, and accessed by a shader within the same draw call. Using it when depth will be written by a draw call or clear command is invalid.

D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE

Value: 0x40

The resource is used with a shader other than the pixel shader. A subresource must be in this state before being read by any stage (except for the pixel shader stage) via a shader resource view. You can still use the resource in a pixel shader with this flag as long as it also has the flag [D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE](#) set. This is a read-only state.

D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE

Value: 0x80

The resource is used with a pixel shader. A subresource must be in this state before being read by the pixel shader via a shader resource view. This is a read-only state.

D3D12_RESOURCE_STATE_STREAM_OUT

Value: 0x100

The resource is used with stream output. A subresource must be in this state when it is accessed by the 3D pipeline as a stream-out target. This is a write-only state.

D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT

Value: 0x200

The resource is used as an indirect argument.

Subresources must be in this state when they are used as the argument buffer passed to the indirect drawing method [ID3D12GraphicsCommandList::ExecuteIndirect](#).

This is a read-only state.

D3D12_RESOURCE_STATE_COPY_DEST

Value: 0x400

The resource is used as the destination in a copy operation.

Subresources must be in this state when they are used as the destination of copy operation, or a blt operation.

This is a write-only state.

D3D12_RESOURCE_STATE_COPY_SOURCE

Value: *0x800*

The resource is used as the source in a copy operation.

Subresources must be in this state when they are used as the source of copy operation, or a blt operation.

This is a read-only state.

D3D12_RESOURCE_STATE_RESOLVE_DEST

Value: *0x1000*

The resource is used as the destination in a resolve operation.

D3D12_RESOURCE_STATE_RESOLVE_SOURCE

Value: *0x2000*

The resource is used as the source in a resolve operation.

D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE

Value: *0x400000*

When a buffer is created with this as its initial state, it indicates that the resource is a raytracing acceleration structure, for use in

[ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure](#),

[ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure](#), or

[ID3D12Device::CreateShaderResourceView](#) for the

[D3D12_SRV_DIMENSION_RAYTRACING_ACCELERATION_STRUCTURE](#) dimension.

(!) Note

A resource to be used for the

D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE state must be created in that state, and then never transitioned out of it. Nor may a resource that was created not in that state be transitioned into it. For more info, see [Acceleration structure memory restrictions](#) in the DirectX raytracing (DXR) functional specification on GitHub.

D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE

Value: *0x1000000*

Starting with Windows 10, version 1903 (10.0; Build 18362), indicates that the resource is a screen-space shading-rate image for variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#).

D3D12_RESOURCE_STATE_GENERIC_READ

D3D12_RESOURCE_STATE_GENERIC_READ is a logically OR'd combination of other read-state bits. This is the required starting state for an upload heap. Your application should generally avoid transitioning to D3D12_RESOURCE_STATE_GENERIC_READ when possible, since that can result in premature cache flushes, or resource layout changes (for example, compress/decompress),

causing unnecessary pipeline stalls. You should instead transition resources only to the actually-used states.

`D3D12_RESOURCE_STATE_ALL_SHADER_RESOURCE`

Equivalent to `D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE` |
`D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE`.

`D3D12_RESOURCE_STATE_PRESENT`

Value: *0*

Synonymous with `D3D12_RESOURCE_STATE_COMMON`.

`D3D12_RESOURCE_STATE_PREDICATION`

Value: *0x200*

The resource is used for [Predication](#).

`D3D12_RESOURCE_STATE_VIDEO_DECODE_READ`

Value: *0x10000*

The resource is used as a source in a decode operation. Examples include reading the compressed bitstream and reading from decode references,

`D3D12_RESOURCE_STATE_VIDEO_DECODE_WRITE`

Value: *0x20000*

The resource is used as a destination in the decode operation. This state is used for decode output and histograms.

`D3D12_RESOURCE_STATE_VIDEO_PROCESS_READ`

Value: *0x40000*

The resource is used to read video data during video processing; that is, the resource is used as the source in a processing operation such as video encoding (compression).

`D3D12_RESOURCE_STATE_VIDEO_PROCESS_WRITE`

Value: *0x80000*

The resource is used to write video data during video processing; that is, the resource is used as the destination in a processing operation such as video encoding (compression).

`D3D12_RESOURCE_STATE_VIDEO_ENCODE_READ`

Value: *0x200000*

The resource is used as the source in an encode operation. This state is used for the input and reference of motion estimation.

`D3D12_RESOURCE_STATE_VIDEO_ENCODE_WRITE`

Value: *0x800000*

This resource is used as the destination in an encode operation. This state is used for the destination texture of a resolve motion vector heap operation.

Remarks

This enum is used by the following methods:

- [CreateCommittedResource](#)
- [CreatePlacedResource](#)
- [CreateReservedResource](#)

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

- [Core Enumerations](#)
- [Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_TRANSITION_BARRIER structure (d3d12.h)

Article02/22/2024

Describes the transition of subresources between different usages.

Syntax

C++

```
typedef struct D3D12_RESOURCE_TRANSITION_BARRIER {
    ID3D12Resource     *pResource;
    UINT                Subresource;
    D3D12_RESOURCE_STATES StateBefore;
    D3D12_RESOURCE_STATES StateAfter;
} D3D12_RESOURCE_TRANSITION_BARRIER;
```

Members

pResource

A pointer to the [ID3D12Resource](#) object that represents the resource used in the transition.

Subresource

The index of the subresource for the transition. Use the [D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES](#) flag (0xffffffff) to transition all subresources in a resource at the same time.

StateBefore

The "before" usages of the subresources, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

StateAfter

The "after" usages of the subresources, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

Remarks

This struct is used by the [Transition](#) member of the [D3D12_RESOURCE_BARRIER](#) struct.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RESOURCE_UAV_BARRIER structure (d3d12.h)

Article02/22/2024

Represents a resource in which all UAV accesses must complete before any future UAV accesses can begin.

Syntax

C++

```
typedef struct D3D12_RESOURCE_UAV_BARRIER {
    ID3D12Resource *pResource;
} D3D12_RESOURCE_UAV_BARRIER;
```

Members

pResource

The resource used in the transition, as a pointer to [ID3D12Resource](#).

Remarks

This struct represents a resource in which all unordered access view (UAV) accesses (reads or writes) must complete before any future UAV accesses (read or write) can begin.

This structure is a member of the [D3D12_RESOURCE_BARRIER](#) structure.

You don't need to insert a UAV barrier between 2 draw or dispatch calls that only read a UAV. Additionally, you don't need to insert a UAV barrier between 2 draw or dispatch calls that write to the same UAV if you know that it's safe to execute the UAV accesses in any order. The resource can be **NULL**, which indicates that any UAV access could require the barrier.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_CONSTANTS structure (d3d12.h)

Article 02/22/2024

Describes constants inline in the root signature that appear in shaders as one constant buffer.

Syntax

C++

```
typedef struct D3D12_ROOT_CONSTANTS {
    UINT ShaderRegister;
    UINT RegisterSpace;
    UINT Num32BitValues;
} D3D12_ROOT_CONSTANTS;
```

Members

ShaderRegister

The shader register.

RegisterSpace

The register space.

Num32BitValues

The number of constants that occupy a single shader slot (these constants appear like a single constant buffer). All constants occupy a single root signature bind slot.

Remarks

Refer to [Resource Binding in HLSL](#) for more information on shader registers and spaces.

D3D12_ROOT_CONSTANTS is the data type of the **Constants** member of **D3D12_ROOT_PARAMETER**. Use a **D3D12_ROOT_CONSTANTS** when you set **D3D12_ROOT_PARAMETER**'s **SlotType** field to the

D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS member of D3D12_ROOT_PARAMETER_TYPE.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_ROOT_CONSTANTS](#)

[Core Structures](#)

[Creating a Root Signature](#)

[Using constants directly in the root signature](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_DESCRIPTOR structure (d3d12.h)

Article02/22/2024

Describes descriptors inline in the root signature version 1.0 that appear in shaders.

Syntax

C++

```
typedef struct D3D12_ROOT_DESCRIPTOR {
    UINT ShaderRegister;
    UINT RegisterSpace;
} D3D12_ROOT_DESCRIPTOR;
```

Members

ShaderRegister

The shader register.

RegisterSpace

The register space.

Remarks

D3D12_ROOT_DESCRIPTOR is the data type of the **Descriptor** member of [D3D12_ROOT_PARAMETER](#). Use a D3D12_ROOT_DESCRIPTOR when you set [D3D12_ROOT_PARAMETER](#)'s **ParameterType** field to the [D3D12_ROOT_PARAMETER_TYPE_CBV](#), [D3D12_ROOT_PARAMETER_TYPE_SRV](#), or [D3D12_ROOT_PARAMETER_TYPE_UAV](#) members of [D3D12_ROOT_PARAMETER_TYPE](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_ROOT_DESCRIPTOR](#)

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR1](#)

[Using descriptors directly in the root signature](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_DESCRIPTOR_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies the volatility of the data referenced by descriptors in a Root Signature 1.1 description, which can enable some driver optimizations.

Syntax

C++

```
typedef enum D3D12_ROOT_DESCRIPTOR_FLAGS {
    D3D12_ROOT_DESCRIPTOR_FLAG_NONE = 0,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_VOLATILE = 0x2,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE = 0x4,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC = 0x8
};
```

Constants

[+] Expand table

D3D12_ROOT_DESCRIPTOR_FLAG_NONE

Value: 0

Default assumptions are made for data (for SRV/CBV: DATA_STATIC_WHILE_SET_AT_EXECUTE, and for UAV: DATA_VOLATILE).

D3D12_ROOT_DESCRIPTOR_FLAG_DATA_VOLATILE

Value: 0x2

Data is volatile. Equivalent to Root Signature Version 1.0.

D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE

Value: 0x4

Data is static while set at execute.

D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC

Value: 0x8

Data is static. The best potential for driver optimization.

Remarks

This enum is used by the [D3D12_ROOT_DESCRIPTOR1](#) structure.

To specify the volatility of both descriptors and data, refer to [D3D12_DESCRIPTOR_RANGE_FLAGS](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_DESCRIPTOR_TABLE structure (d3d12.h)

Article02/22/2024

Describes the root signature 1.0 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

Syntax

C++

```
typedef struct D3D12_ROOT_DESCRIPTOR_TABLE {
    UINT NumDescriptorRanges;
    const D3D12_DESCRIPTOR_RANGE *pDescriptorRanges;
} D3D12_ROOT_DESCRIPTOR_TABLE;
```

Members

NumDescriptorRanges

The number of descriptor ranges in the table layout.

pDescriptorRanges

An array of [D3D12_DESCRIPTOR_RANGE](#) structures that describe the descriptor ranges.

Remarks

Samplers are not allowed in the same descriptor table as constant-buffer views (CBVs), unordered-access views (UAVs), and shader-resource views (SRVs).

`D3D12_ROOT_DESCRIPTOR_TABLE` is the data type of the `DescriptorTable` member of `D3D12_ROOT_PARAMETER`. Use a `D3D12_ROOT_DESCRIPTOR_TABLE` when you set `D3D12_ROOT_PARAMETER`'s `ParameterType` member to [D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_ROOT_DESCRIPTOR_TABLE](#)

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR_TABLE1](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_DESCRIPTOR_TABLE1 structure (d3d12.h)

Article02/22/2024

Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that are all relative to a single base descriptor handle.

Syntax

C++

```
typedef struct D3D12_ROOT_DESCRIPTOR_TABLE1 {
    UINT NumDescriptorRanges;
    const D3D12_DESCRIPTOR_RANGE1 *pDescriptorRanges;
} D3D12_ROOT_DESCRIPTOR_TABLE1;
```

Members

NumDescriptorRanges

The number of descriptor ranges in the table layout.

pDescriptorRanges

An array of [D3D12_DESCRIPTOR_RANGE1](#) structures that describe the descriptor ranges.

Remarks

Samplers are not allowed in the same descriptor table as constant-buffer views (CBVs), unordered-access views (UAVs), and shader-resource views (SRVs).

`D3D12_ROOT_DESCRIPTOR_TABLE1` is the data type of the `DescriptorTable` member of `D3D12_ROOT_PARAMETER1`. Use a `D3D12_ROOT_DESCRIPTOR_TABLE1` when you set `D3D12_ROOT_PARAMETER1`'s `SlotType` member to [D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE](#).

Refer to the helper structure [CD3DX12_ROOT_DESCRIPTOR_TABLE1](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR_TABLE](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_DESCRIPTOR1 structure (d3d12.h)

Article 02/22/2024

Describes descriptors inline in the root signature version 1.1 that appear in shaders.

Syntax

C++

```
typedef struct D3D12_ROOT_DESCRIPTOR1 {
    UINT ShaderRegister;
    UINT RegisterSpace;
    D3D12_ROOT_DESCRIPTOR_FLAGS Flags;
} D3D12_ROOT_DESCRIPTOR1;
```

Members

ShaderRegister

The shader register.

RegisterSpace

The register space.

Flags

Specifies the [D3D12_ROOT_DESCRIPTOR_FLAGS](#) that determine the volatility of descriptors and the data they reference.

Remarks

D3D12_ROOT_DESCRIPTOR1 is the data type of the **Descriptor** member of [D3D12_ROOT_PARAMETER1](#). Use a D3D12_ROOT_DESCRIPTOR1 when you set [D3D12_ROOT_PARAMETER1](#)'s **ParameterType** field to the D3D12_ROOT_PARAMETER_TYPE_CBV, D3D12_ROOT_PARAMETER_TYPE_SRV, or D3D12_ROOT_PARAMETER_TYPE_UAV members of [D3D12_ROOT_PARAMETER_TYPE](#).

Refer to the helper structure [CD3DX12_ROOT_DESCRIPTOR1](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_PARAMETER structure (d3d12.h)

Article04/02/2021

Describes the slot of a root signature version 1.0.

Syntax

C++

```
typedef struct D3D12_ROOT_PARAMETER {
    D3D12_ROOT_PARAMETER_TYPE ParameterType;
    union {
        D3D12_ROOT_DESCRIPTOR_TABLE DescriptorTable;
        D3D12_ROOT_CONSTANTS        Constants;
        D3D12_ROOT_DESCRIPTOR        Descriptor;
    };
    D3D12_SHADER_VISIBILITY     ShaderVisibility;
} D3D12_ROOT_PARAMETER;
```

Members

ParameterType

A [D3D12_ROOT_PARAMETER_TYPE](#)-typed value that specifies the type of root signature slot. This member determines which type to use in the union below.

DescriptorTable

A [D3D12_ROOT_DESCRIPTOR_TABLE](#) structure that describes the layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.

Constants

A [D3D12_ROOT_CONSTANTS](#) structure that describes constants inline in the root signature that appear in shaders as one constant buffer.

Descriptor

A [D3D12_ROOT_DESCRIPTOR](#) structure that describes descriptors inline in the root signature that appear in shaders.

ShaderVisibility

A [D3D12_SHADER_VISIBILITY](#)-typed value that specifies the shaders that can access the contents of the root signature slot.

Remarks

A [D3D12_ROOT_SIGNATURE_DESC](#) can contain descriptor tables and inline constants. More capable hardware could support inline descriptors in the root signature as well. The number of bind slots in the root signature are most efficient if kept below a certain size, and can have an upper bound as well.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_ROOT_PARAMETER](#)

[Core Structures](#)

[Creating a Root Signature](#)

[D3D12_ROOT_PARAMETER1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_PARAMETER_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of root signature slot.

Syntax

C++

```
typedef enum D3D12_ROOT_PARAMETER_TYPE {
    D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE = 0,
    D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS,
    D3D12_ROOT_PARAMETER_TYPE_CBV,
    D3D12_ROOT_PARAMETER_TYPE_SRV,
    D3D12_ROOT_PARAMETER_TYPE_UAV
};
```

Constants

[+] Expand table

D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE

Value: 0

The slot is for a descriptor table.

D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS

The slot is for root constants.

D3D12_ROOT_PARAMETER_TYPE_CBV

The slot is for a constant-buffer view (CBV).

D3D12_ROOT_PARAMETER_TYPE_SRV

The slot is for a shader-resource view (SRV).

D3D12_ROOT_PARAMETER_TYPE_UAV

The slot is for a unordered-access view (UAV).

Remarks

This enum is used by the [D3D12_ROOT_PARAMETER](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Creating a Root Signature](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_PARAMETER1 structure (d3d12.h)

Article 02/22/2024

Describes the slot of a root signature version 1.1.

Syntax

C++

```
typedef struct D3D12_ROOT_PARAMETER1 {
    D3D12_ROOT_PARAMETER_TYPE ParameterType;
    union {
        D3D12_ROOT_DESCRIPTOR_TABLE1 DescriptorTable;
        D3D12_ROOT_CONSTANTS          Constants;
        D3D12_ROOT_DESCRIPTOR1         Descriptor;
    };
    D3D12_SHADER_VISIBILITY      ShaderVisibility;
} D3D12_ROOT_PARAMETER1;
```

Members

ParameterType

A [D3D12_ROOT_PARAMETER_TYPE](#)-typed value that specifies the type of root signature slot. This member determines which type to use in the union below.

DescriptorTable

A [D3D12_ROOT_DESCRIPTOR_TABLE1](#) structure that describes the layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.

Constants

A [D3D12_ROOT_CONSTANTS](#) structure that describes constants inline in the root signature that appear in shaders as one constant buffer.

Descriptor

A [D3D12_ROOT_DESCRIPTOR1](#) structure that describes descriptors inline in the root signature that appear in shaders.

ShaderVisibility

A [D3D12_SHADER_VISIBILITY](#)-typed value that specifies the shaders that can access the contents of the root signature slot.

Remarks

Use this structure with the [D3D12_ROOT_SIGNATURE_DESC1](#) structure.

Refer to the helper structure [CD3DX12_ROOT_PARAMETER1](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_PARAMETER](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_SIGNATURE_DESC structure (d3d12.h)

Article02/22/2024

Describes the layout of a root signature version 1.0.

Syntax

C++

```
typedef struct D3D12_ROOT_SIGNATURE_DESC {
    UINT NumParameters;
    const D3D12_ROOT_PARAMETER *pParameters;
    UINT NumStaticSamplers;
    const D3D12_STATIC_SAMPLER_DESC *pStaticSamplers;
    D3D12_ROOT_SIGNATURE_FLAGS Flags;
} D3D12_ROOT_SIGNATURE_DESC;
```

Members

NumParameters

The number of slots in the root signature. This number is also the number of elements in the *pParameters* array.

pParameters

An array of [D3D12_ROOT_PARAMETER](#) structures for the slots in the root signature.

NumStaticSamplers

Specifies the number of static samplers.

pStaticSamplers

Pointer to one or more [D3D12_STATIC_SAMPLER_DESC](#) structures.

Flags

A combination of [D3D12_ROOT_SIGNATURE_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for the root signature layout.

Remarks

This structure is used by the [D3D12SerializeRootSignature](#) function and is returned by the [ID3D12RootSignatureDeserializer::GetRootSignatureDesc](#) method.

There is one graphics root signature, and one compute root signature.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_ROOT_SIGNATURE_DESC](#)

[Core Structures](#)

[Creating a Root Signature](#)

[D3D12_ROOT_PARAMETER_TYPE](#)

[D3D12_ROOT_SIGNATURE_DESC1](#)

[D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#)

[Using constants directly in the root signature](#)

[Using descriptors directly in the root signature](#)

Feedback

Was this page helpful?

[] Yes

[] No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_SIGNATURE_DESC1 structure (d3d12.h)

Article 02/22/2024

Describes the layout of a root signature version 1.1.

Syntax

C++

```
typedef struct D3D12_ROOT_SIGNATURE_DESC1 {
    UINT NumParameters;
    const D3D12_ROOT_PARAMETER1 *pParameters;
    UINT NumStaticSamplers;
    const D3D12_STATIC_SAMPLER_DESC *pStaticSamplers;
    D3D12_ROOT_SIGNATURE_FLAGS Flags;
} D3D12_ROOT_SIGNATURE_DESC1;
```

Members

NumParameters

The number of slots in the root signature. This number is also the number of elements in the *pParameters* array.

pParameters

An array of [D3D12_ROOT_PARAMETER1](#) structures for the slots in the root signature.

NumStaticSamplers

Specifies the number of static samplers.

pStaticSamplers

Pointer to one or more [D3D12_STATIC_SAMPLER_DESC](#) structures.

Flags

Specifies the [D3D12_ROOT_SIGNATURE_FLAGS](#) that determine the data volatility.

Remarks

Use this structure with the [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#) structure.

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_SIGNATURE_DESC](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_ROOT_SIGNATURE_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies options for root signature layout.

Syntax

C++

```
typedef enum D3D12_ROOT_SIGNATURE_FLAGS {
    D3D12_ROOT_SIGNATURE_FLAG_NONE = 0,
    D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT = 0x1,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_VERTEX_SHADER_ROOT_ACCESS = 0x2,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_HULL_SHADER_ROOT_ACCESS = 0x4,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_DOMAIN_SHADER_ROOT_ACCESS = 0x8,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_GEOMETRY_SHADER_ROOT_ACCESS = 0x10,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_PIXEL_SHADER_ROOT_ACCESS = 0x20,
    D3D12_ROOT_SIGNATURE_FLAG_ALLOW_STREAM_OUTPUT = 0x40,
    D3D12_ROOT_SIGNATURE_FLAG_LOCAL_ROOT_SIGNATURE = 0x80,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_AMPLIFICATION_SHADER_ROOT_ACCESS = 0x100,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_MESH_SHADER_ROOT_ACCESS = 0x200,
    D3D12_ROOT_SIGNATURE_FLAG_CBV_SRV_UAV_HEAP_DIRECTLY_INDEXED = 0x400,
    D3D12_ROOT_SIGNATURE_FLAG_SAMPLER_HEAP_DIRECTLY_INDEXED = 0x800
} ;
```

Constants

[] Expand table

`D3D12_ROOT_SIGNATURE_FLAG_NONE`

Value: *0*

Indicates default behavior.

`D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT`

Value: *0x1*

The app is opting in to using the Input Assembler (requiring an input layout that defines a set of vertex buffer bindings). Omitting this flag can result in one root argument space being saved on some hardware. Omit this flag if the Input Assembler is not required, though the optimization is minor.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_VERTEX_SHADER_ROOT_ACCESS`

Value: `0x2`

Denies the vertex shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_HULL_SHADER_ROOT_ACCESS`

Value: `0x4`

Denies the hull shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_DOMAIN_SHADER_ROOT_ACCESS`

Value: `0x8`

Denies the domain shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_GEOMETRY_SHADER_ROOT_ACCESS`

Value: `0x10`

Denies the geometry shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_PIXEL_SHADER_ROOT_ACCESS`

Value: `0x20`

Denies the pixel shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_ALLOW_STREAM_OUTPUT`

Value: `0x40`

The app is opting in to using Stream Output. Omitting this flag can result in one root argument space being saved on some hardware. Omit this flag if Stream Output is not required, though the optimization is minor.

`D3D12_ROOT_SIGNATURE_FLAG_LOCAL_ROOT_SIGNATURE`

Value: `0x80`

The root signature is to be used with raytracing shaders to define resource bindings sourced from shader records in shader tables. This flag cannot be combined with any other root signature flags, which are all related to the graphics pipeline. The absence of the flag means the root signature can be used with graphics or compute, where the compute version is also shared with raytracing's global root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_AMPLIFICATION_SHADER_ROOT_ACCESS`

Value: `0x100`

Denies the amplification shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_DENY_MESH_SHADER_ROOT_ACCESS`

Value: `0x200`

Denies the mesh shader access to the root signature.

`D3D12_ROOT_SIGNATURE_FLAG_CBV_SRV_UAV_HEAP_DIRECTLY_INDEXED`

Value: `0x400`

The shaders are allowed to index the CBV/SRV/UAV descriptor heap directly, using the *ResourceDescriptorHeap* built-in variable.

`D3D12_ROOT_SIGNATURE_FLAG_SAMPLER_HEAP_DIRECTLY_INDEXED`

Value: `0x800`

The shaders are allowed to index the sampler descriptor heap directly, using the `SamplerDescriptorHeap` built-in variable.

Remarks

This enum is used in the [D3D12_ROOT_SIGNATURE_DESC](#) structure.

The value in denying access to shader stages is a minor optimization on some hardware. If, for example, the [D3D12_SHADER_VISIBILITY_ALL](#) flag has been set to broadcast the root signature to all shader stages, then denying access can overrule this and save the hardware some work. Alternatively if the shader is so simple that no root signature resources are needed, then denying access could be used here too.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[Creating a Root Signature](#)

[D3D12_ROOT_SIGNATURE_DESC](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RT_FORMAT_ARRAY structure (d3d12.h)

Article 02/22/2024

Wraps an array of render target formats.

Syntax

C++

```
struct D3D12_RT_FORMAT_ARRAY {
    DXGI_FORMAT RTFormats[8];
    UINT        NumRenderTargets;
};
```

Members

RTFormats[8]

Specifies a fixed-size array of DXGI_FORMAT values that define the format of up to 8 render targets.

NumRenderTargets

Specifies the number of render target formats stored in the array.

Remarks

This structure is primarily intended to be used when creating pipeline state stream descriptions that contain multiple contiguous render target format descriptions.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_RTV_DIMENSION enumeration (d3d12.h)

Article02/22/2024

Identifies the type of resource to view as a render target.

Syntax

C++

```
typedef enum D3D12_RTV_DIMENSION {
    D3D12_RTV_DIMENSION_UNKNOWN = 0,
    D3D12_RTV_DIMENSION_BUFFER = 1,
    D3D12_RTV_DIMENSION_TEXTURE1D = 2,
    D3D12_RTV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D12_RTV_DIMENSION_TEXTURE2D = 4,
    D3D12_RTV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D12_RTV_DIMENSION_TEXTURE2DMS = 6,
    D3D12_RTV_DIMENSION_TEXTURE2DMSARRAY = 7,
    D3D12_RTV_DIMENSION_TEXTURE3D = 8
};
```

Constants

[] Expand table

D3D12_RTV_DIMENSION_UNKNOWN

Value: 0

Do not use this value, as it will cause [ID3D12Device::CreateRenderTargetView](#) to fail.

D3D12_RTV_DIMENSION_BUFFER

Value: 1

The resource will be accessed as a buffer.

D3D12_RTV_DIMENSION_TEXTURE1D

Value: 2

The resource will be accessed as a 1D texture.

D3D12_RTV_DIMENSION_TEXTURE1DARRAY

Value: 3

The resource will be accessed as an array of 1D textures.

`D3D12_RTV_DIMENSION_TEXTURE2D`

Value: 4

The resource will be accessed as a 2D texture.

`D3D12_RTV_DIMENSION_TEXTURE2DARRAY`

Value: 5

The resource will be accessed as an array of 2D textures.

`D3D12_RTV_DIMENSION_TEXTURE2DMS`

Value: 6

The resource will be accessed as a 2D texture with multisampling.

`D3D12_RTV_DIMENSION_TEXTURE2DMSARRAY`

Value: 7

The resource will be accessed as an array of 2D textures with multisampling.

`D3D12_RTV_DIMENSION_TEXTURE3D`

Value: 8

The resource will be accessed as a 3D texture.

Remarks

Specify one of the values in this enumeration in the `ViewDimension` member of a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_SAMPLE_POSITION structure (d3d12.h)

Article04/02/2021

Describes a sub-pixel sample position for use with programmable sample positions.

Syntax

C++

```
typedef struct D3D12_SAMPLE_POSITION {
    INT8 X;
    INT8 Y;
} D3D12_SAMPLE_POSITION;
```

Members

X

A signed sub-pixel coordinate value in the X axis.

Y

A signed sub-pixel coordinate value in the Y axis.

Remarks

Sample positions have the origin (0, 0) at the pixel center. Each of the X and Y coordinates are signed values in the range -8 (top/left) to 7 (bottom/right). Values outside this range are invalid.

Each increment of these integer values represents 1/16th of a pixel. For example, X and Y values of -8 and 4, respectively, correspond to floating-point values of -0.5 and 0.25, a point located on the left-most edge of the pixel, half-way between its center-line and the bottom edge. Because of this, the bottom-most and right-most edge of a pixel are not reachable by sampling (these positions are reachable as the top-most and left-most edges of the neighboring pixels).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SAMPLER_DESC structure (d3d12.h)

Article 02/22/2024

Describes a sampler state.

Syntax

C++

```
typedef struct D3D12_SAMPLER_DESC {
    D3D12_FILTER             Filter;
    D3D12_TEXTURE_ADDRESS_MODE AddressU;
    D3D12_TEXTURE_ADDRESS_MODE AddressV;
    D3D12_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT                     MipLODBias;
    UINT                      MaxAnisotropy;
    D3D12_COMPARISON_FUNC     ComparisonFunc;
    FLOAT                     BorderColor[4];
    FLOAT                     MinLOD;
    FLOAT                     MaxLOD;
} D3D12_SAMPLER_DESC;
```

Members

Filter

A [D3D12_FILTER](#)-typed value that specifies the filtering method to use when sampling a texture.

AddressU

A [D3D12_TEXTURE_ADDRESS_MODE](#)-typed value that specifies the method to use for resolving a u texture coordinate that is outside the 0 to 1 range.

AddressV

A [D3D12_TEXTURE_ADDRESS_MODE](#)-typed value that specifies the method to use for resolving a v texture coordinate that is outside the 0 to 1 range.

AddressW

A [D3D12_TEXTURE_ADDRESS_MODE](#)-typed value that specifies the method to use for resolving a w texture coordinate that is outside the 0 to 1 range.

MipLODBias

Offset from the calculated mipmap level. For example, if the runtime calculates that a texture should be sampled at mipmap level 3 and **MipLODBias** is 2, the texture will be sampled at mipmap level 5.

MaxAnisotropy

Clamping value used if [D3D12_FILTER_ANISOTROPIC](#) or [D3D12_FILTER_COMPARISON_ANISOTROPIC](#) is specified in **Filter**. Valid values are between 1 and 16.

ComparisonFunc

A [D3D12_COMPARISON_FUNC](#)-typed value that specifies a function that compares sampled data against existing sampled data.

BorderColor[4]

RGBA border color to use if [D3D12_TEXTURE_ADDRESS_MODE_BORDER](#) is specified for **AddressU**, **AddressV**, or **AddressW**. Range must be between 0.0 and 1.0 inclusive.

MinLOD

Lower end of the mipmap range to clamp access to, where 0 is the largest and most detailed mipmap level and any level higher than that is less detailed.

MaxLOD

Upper end of the mipmap range to clamp access to, where 0 is the largest and most detailed mipmap level and any level higher than that is less detailed. This value must be greater than or equal to **MinLOD**. To have no upper limit on LOD, set this member to a large value.

Remarks

This structure is used by [CreateSampler](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SAMPLER_FEEDBACK_TIER enumeration (d3d12.h)

Article02/27/2023

Defines constants that specify sampler feedback support.

Syntax

C++

```
typedef enum D3D12_SAMPLER_FEEDBACK_TIER {
    D3D12_SAMPLER_FEEDBACK_TIER_NOT_SUPPORTED = 0,
    D3D12_SAMPLER_FEEDBACK_TIER_0_9 = 90,
    D3D12_SAMPLER_FEEDBACK_TIER_1_0 = 100
} ;
```

Constants

 Expand table

`D3D12_SAMPLER_FEEDBACK_TIER_NOT_SUPPORTED`

Value: 0

Specifies that sampler feedback is not supported. Attempts at calling sampler feedback APIs represent an error.

`D3D12_SAMPLER_FEEDBACK_TIER_0_9`

Value: 90

Specifies that sampler feedback is supported to tier 0.9. This indicates the following:

Sampler feedback is supported for samplers with these texture addressing modes:

- * `D3D12_TEXTURE_ADDRESS_MODE_WRAP`
- * `D3D12_TEXTURE_ADDRESS_MODE_CLAMP`

The Texture2D shader resource view passed in to feedback-writing HLSL methods has these restrictions:

- * The `MostDetailedMip` field must be 0.
- * The `MipLevels` count must span the full mip count of the resource.
- * The `PlaneSlice` field must be 0.
- * The `ResourceMinLODClamp` field must be 0.

The Texture2DArray shader resource view passed in to feedback-writing HLSL methods has these

restrictions:

- * All the limitations as in Texture2D above, and
- * The FirstArraySlice field must be 0.
- * The ArraySize field must span the full array element count of the resource.

D3D12_SAMPLER_FEEDBACK_TIER_1_0

Value: 100

Specifies sample feedback is supported to tier 1.0. This indicates that sampler feedback is supported for all texture addressing modes, and feedback-writing methods are supported irrespective of the passed-in shader resource view.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Sampler feedback spec ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER structure (d3d12.h)

Article 02/22/2024

Opaque data structure describing driver versioning for a serialized acceleration structure. Pass this structure into a call to [ID3D12Device5::CheckDriverMatchingIdentifier](#) to determine if a previously serialized acceleration structure is compatible with the current driver/device, and can therefore be deserialized and used for raytracing.

Syntax

C++

```
typedef struct D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER {
    GUID DriverOpaqueGUID;
    BYTE DriverOpaqueVersioningData[16];
} D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER;
```

Members

`DriverOpaqueGUID`

The opaque identifier of the driver.

`DriverOpaqueVersioningData[16]`

The opaque driver versioning data.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_SERIALIZED_DATA_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies the type of serialized data. Use a value from this enumeration when calling [ID3D12Device5::CheckDriverMatchingIdentifier](#).

Syntax

C++

```
typedef enum D3D12_SERIALIZED_DATA_TYPE {
    D3D12_SERIALIZED_DATA_RAYTRACING_ACCELERATION_STRUCTURE = 0
};
```

Constants

 Expand table

D3D12_SERIALIZED_DATA_RAYTRACING_ACCELERATION_STRUCTURE

Value: 0

The serialized data is a raytracing acceleration structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER structure (d3d12.h)

Article 02/22/2024

Defines the header for a serialized raytracing acceleration structure.

Syntax

C++

```
typedef struct D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER {  
    D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER DriverMatchingIdentifier;  
    UINT64  
    SerializedSizeInBytesIncludingHeader;  
    UINT64                                DeserializedSizeInBytes;  
    UINT64  
    NumBottomLevelAccelerationStructurePointersAfterHeader;  
} D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER;
```

Members

`DriverMatchingIdentifier`

The driver-matching identifier.

`SerializedSizeInBytesIncludingHeader`

The size of serialized data.

`DeserializedSizeInBytes`

Size of the memory that will be consumed when the acceleration structure is deserialized. This value is less than or equal to the size of the original acceleration structure before it was serialized.

`NumBottomLevelAccelerationStructurePointersAfterHeader`

Size of the array of `D3D12_GPU_VIRTUAL_ADDRESS` values that follow the header. For more information, see

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_D
ESC.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_BYTECODE structure (d3d12.h)

Article 02/22/2024

Describes shader data.

Syntax

C++

```
typedef struct D3D12_SHADER_BYTECODE {
    const void *pShaderBytecode;
    SIZE_T BytecodeLength;
} D3D12_SHADER_BYTECODE;
```

Members

pShaderBytecode

A pointer to a memory block that contains the shader data.

BytecodeLength

The size, in bytes, of the shader data that the **pShaderBytecode** member points to.

Remarks

The [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) objects contain **D3D12_SHADER_BYTECODE** structures that describe various shader types.

When loading a shader from FXC/DXC, this may be the entire compiled blob as is loaded from disk.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_SHADER_BYTECODE](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_SHADER_CACHE_CONTROL_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify shader cache control options.

Syntax

C++

```
typedef enum D3D12_SHADER_CACHE_CONTROL_FLAGS {
    D3D12_SHADER_CACHE_CONTROL_FLAG_DISABLE = 0x1,
    D3D12_SHADER_CACHE_CONTROL_FLAG_ENABLE = 0x2,
    D3D12_SHADER_CACHE_CONTROL_FLAG_CLEAR = 0x4
};
```

Constants

[+] Expand table

`D3D12_SHADER_CACHE_CONTROL_FLAG_DISABLE`

Value: `0x1`

Specifies that the cache shouldn't be used to look up data, and shouldn't have new data stored in it. Attempts to use/create a cache while it's disabled result in `DXGI_ERROR_NOT_CURRENTLY_AVAILABLE`.

`D3D12_SHADER_CACHE_CONTROL_FLAG_ENABLE`

Value: `0x2`

Specifies that use of the cache should be resumed.

`D3D12_SHADER_CACHE_CONTROL_FLAG_CLEAR`

Value: `0x4`

Specifies that any existing contents of the cache should be deleted.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [D3D12 shader cache APIs ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_CACHE_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify shader cache flags.

Syntax

C++

```
typedef enum D3D12_SHADER_CACHE_FLAGS {
    D3D12_SHADER_CACHE_FLAG_NONE = 0,
    D3D12_SHADER_CACHE_FLAG_DRIVER_VERSIONED = 0x1,
    D3D12_SHADER_CACHE_FLAG_USE_WORKING_DIR = 0x2
};
```

Constants

[+] [Expand table](#)

<code>D3D12_SHADER_CACHE_FLAG_NONE</code>

Value: `0`

Specifies no flag.

<code>D3D12_SHADER_CACHE_FLAG_DRIVER_VERSIONED</code>

Value: `0x1`

Specifies that the cache is implicitly versioned by the driver being used. For multi-GPU systems, a cache created this way is stored side by side for each adapter on which the application runs. The *Version* field in the [D3D12_SHADER_CACHE_SESSION_DESC](#) struct (the cache description) is used as an additional constraint.

<code>D3D12_SHADER_CACHE_FLAG_USE_WORKING_DIR</code>
--

Value: `0x2`

By default, caches are stored in temporary storage, and can be cleared by disk cleanup. This constant (not valid for UWP apps) specifies that the cache is instead stored in the current working directory.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [D3D12_SHADER_CACHE_SESSION_DESC](#)
- [D3D12 shader cache APIs ↗](#)

Feedback

Was this page helpful?

[!\[\]\(e94f983b0f483d7e430f5009cd46898d_img.jpg\) Yes](#)[!\[\]\(e83e501476cf25182513b639be4d7362_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_CACHE_KIND_FLAGS enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a kind of shader cache.

Syntax

C++

```
typedef enum D3D12_SHADER_CACHE_KIND_FLAGS {
    D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_D3D_CACHE_FOR_DRIVER = 0x1,
    D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_D3D_CONVERSIONS = 0x2,
    D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_DRIVER_MANAGED = 0x4,
    D3D12_SHADER_CACHE_KIND_FLAG_APPLICATION_MANAGED = 0x8
} ;
```

Constants

[+] Expand table

D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_D3D_CACHE_FOR_DRIVER

Value: 0x1

Specifies a cache that's managed by Direct3D 12 to store driver compilations of application shaders.

D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_D3D_CONVERSIONS

Value: 0x2

Specifies a cache that's used to store Direct3D 12's conversions of one shader type to another (for example, DXBC shaders to DXIL shaders).

D3D12_SHADER_CACHE_KIND_FLAG_IMPLICIT_DRIVER_MANAGED

Value: 0x4

Specifies a cache that's managed by the driver. Operations for this cache are hints.

D3D12_SHADER_CACHE_KIND_FLAG_APPLICATION_MANAGED

Value: 0x8

Specifies all shader cache sessions created by the [ID3D12Device9::CreateShaderCacheSession](#) method. Requests to **CLEAR** with this flag apply to all currently active application cache sessions, as well as on-disk caches created without [D3D12_SHADER_CACHE_FLAG_USE_WORKING_DIR](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [D3D12 shader cache APIs](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_CACHE_MODE enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a shader cache's mode.

Syntax

C++

```
typedef enum D3D12_SHADER_CACHE_MODE {
    D3D12_SHADER_CACHE_MODE_MEMORY = 0,
    D3D12_SHADER_CACHE_MODE_DISK
};
```

Constants

[+] Expand table

`D3D12_SHADER_CACHE_MODE_MEMORY`

Value: 0

Specifies that there's no backing file for this cache. All stores are discarded when the session object is destroyed.

`D3D12_SHADER_CACHE_MODE_DISK`

Specifies that the session is backed by files on disk that persist from run to run unless cleared. For ways to clear a disk cache, see [ID3D12ShaderCacheSession::SetDeleteOnDestroy](#).

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [ID3D12ShaderCacheSession::SetDeleteOnDestroy](#)
 - [D3D12 shader cache APIs ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_CACHE_SESSION_DESC structure (d3d12.h)

Article02/22/2024

Describes a shader cache session.

Syntax

C++

```
typedef struct D3D12_SHADER_CACHE_SESSION_DESC {
    GUID Identifier;
    D3D12_SHADER_CACHE_MODE Mode;
    D3D12_SHADER_CACHE_FLAGS Flags;
    UINT MaximumInMemoryCacheSizeBytes;
    UINT MaximumInMemoryCacheEntries;
    UINT MaximumValueFileSizeBytes;
    UINT64 Version;
} D3D12_SHADER_CACHE_SESSION_DESC;
```

Members

Identifier

Type: [GUID](#)

A unique identifier to give to this specific cache. Caches with different identifiers are stored side by side. Caches with the same identifier are shared across all sessions in the same process. Creating a disk cache with the same identifier as an already-existing cache opens that cache, unless the **Version** doesn't matches. In that case, if there are no other sessions open to that cache, it is cleared and re-created. If there are existing sessions, then [ID3D12Device9::CreateShaderCacheSession](#) returns **DXGI_ERROR_ALREADY_EXISTS**.

Mode

Type: [D3D12_SHADER_CACHE_MODE](#)

Specifies the kind of cache.

Flags

Type: [D3D12_SHADER_CACHE_FLAGS](#)

Modifies the behavior of the cache.

`MaximumInMemoryCacheSizeBytes`

Type: [UINT](#)

For in-memory caches, this is the only storage available. For disk caches, all entries that are stored or found are temporarily stored in memory, until evicted by newer entries. This value determines the size of that temporary storage. Defaults to 1KB.

`MaximumInMemoryCacheEntries`

Type: [UINT](#)

Specifies how many entries can be stored in memory. Defaults to 128.

`MaximumValueFileSizeBytes`

Type: [UINT](#)

For disk caches, controls the maximum file size. Defaults to 128MB.

`Version`

Type: [UINT64](#)

Can be used to implicitly clear caches when an application or component update is done. If the version doesn't match the version stored in the cache, then it will be wiped and re-created.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- ID3D12Device9::CreateShaderCacheSession
 - D3D12 shader cache APIs ↗
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_CACHE_SUPPORT_FLAG

S enumeration (d3d12.h)

Article 02/22/2024

Describes the level of support for shader caching in the current graphics driver.

Syntax

C++

```
typedef enum D3D12_SHADER_CACHE_SUPPORT_FLAGS {
    D3D12_SHADER_CACHE_SUPPORT_NONE = 0,
    D3D12_SHADER_CACHE_SUPPORT_SINGLE_PSO = 0x1,
    D3D12_SHADER_CACHE_SUPPORT_LIBRARY = 0x2,
    D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_INPROC_CACHE = 0x4,
    D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_DISK_CACHE = 0x8,
    D3D12_SHADER_CACHE_SUPPORT_DRIVER_MANAGED_CACHE,
    D3D12_SHADER_CACHE_SUPPORT_SHADER_CONTROL_CLEAR,
    D3D12_SHADER_CACHE_SUPPORT_SHADER_SESSION_DELETE
} ;
```

Constants

[+] Expand table

D3D12_SHADER_CACHE_SUPPORT_NONE

Value: 0

Indicates that the driver does not support shader caching.

D3D12_SHADER_CACHE_SUPPORT_SINGLE_PSO

Value: 0x1

Indicates that the driver supports the CachedPSO member of the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structures. This is always supported.

D3D12_SHADER_CACHE_SUPPORT_LIBRARY

Value: 0x2

Indicates that the driver supports the ID3D12PipelineLibrary interface, which provides application-controlled PSO grouping and caching. This is supported by drivers targetting the Windows 10 Anniversary Update.

`D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_INPROC_CACHE`

Value: *0x4*

Indicates that the driver supports an OS-managed shader cache that stores compiled shaders in memory during the current run of the application.

`D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_DISK_CACHE`

Value: *0x8*

Indicates that the driver supports an OS-managed shader cache that stores compiled shaders on disk to accelerate future runs of the application.

Remarks

This enum is used by the [D3D_FEATURE_DATA_SHADER_CACHE](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_COMPONENT_MAPPING enumeration (d3d12.h)

Article 01/31/2022

Specifies how memory gets routed by a shader resource view (SRV).

Syntax

C++

```
typedef enum D3D12_SHADER_COMPONENT_MAPPING {
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_0 = 0,
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_1 = 1,
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_2 = 2,
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3 = 3,
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0 = 4,
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1 = 5
};
```

Constants

[Expand table](#)

D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_0

Value: 0

Indicates return component 0 (red).

D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_1

Value: 1

Indicates return component 1 (green).

D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_2

Value: 2

Indicates return component 2 (blue).

D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3

Value: 3

Indicates return component 3 (alpha).

D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0

Value: 4

Indicates forcing the resulting value to 0.

D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1

Value: 5

Indicates forcing the resulting value 1. The value of forcing 1 is either 0x1 or 1.0f depending on the format type for that component in the source format.

Remarks

This enum allows the SRV to select how memory gets routed to the four return components in a shader after a memory fetch. The options for each shader component [0..3] (corresponding to RGBA) are: component 0..3 from the SRV fetch result or force 0 or force 1.

The default 1:1 mapping can be indicated by specifying

D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING, otherwise an arbitrary mapping can be specified using the macro

D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING.

See below.

Note the following defines.

C++

```
#define D3D12_SHADER_COMPONENT_MAPPING_MASK 0x7
#define D3D12_SHADER_COMPONENT_MAPPING_SHIFT 3
#define
D3D12_SHADER_COMPONENT_MAPPING_ALWAYS_SET_BIT_AVOIDING_ZEROMEM_MISTAKES \
    (1<<(D3D12_SHADER_COMPONENT_MAPPING_SHIFT*4))
#define D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING(Src0,Src1,Src2,Src3) \
    (((Src0)&D3D12_SHADER_COMPONENT_MAPPING_MASK)| \
     (((Src1)&D3D12_SHADER_COMPONENT_MAPPING_MASK) \
      <<D3D12_SHADER_COMPONENT_MAPPING_SHIFT)| \
     (((Src2)&D3D12_SHADER_COMPONENT_MAPPING_MASK) \
      <<(D3D12_SHADER_COMPONENT_MAPPING_SHIFT*2))| \
     (((Src3)&D3D12_SHADER_COMPONENT_MAPPING_MASK) \
      <<(D3D12_SHADER_COMPONENT_MAPPING_SHIFT*3))| \
     D3D12_SHADER_COMPONENT_MAPPING_ALWAYS_SET_BIT_AVOIDING_ZEROMEM_MISTAKES)
#define D3D12_DECODE_SHADER_4_COMPONENT_MAPPING(ComponentToExtract,Mapping) \
    ((D3D12_SHADER_COMPONENT_MAPPING)(Mapping >>
(D3D12_SHADER_COMPONENT_MAPPING_SHIFT*ComponentToExtract) &
D3D12_SHADER_COMPONENT_MAPPING_MASK))
```

```
#define D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING  
D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING(0,1,2,3)
```

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_MIN_PRECISION_SUPPORT enumeration (d3d12.h)

Article 02/22/2024

Describes minimum precision support options for shaders in the current graphics driver.

Syntax

C++

```
typedef enum D3D12_SHADER_MIN_PRECISION_SUPPORT {
    D3D12_SHADER_MIN_PRECISION_SUPPORT_NONE = 0,
    D3D12_SHADER_MIN_PRECISION_SUPPORT_10_BIT = 0x1,
    D3D12_SHADER_MIN_PRECISION_SUPPORT_16_BIT = 0x2
};
```

Constants

[] Expand table

<code>D3D12_SHADER_MIN_PRECISION_SUPPORT_NONE</code>
--

Value: *0*

The driver supports only full 32-bit precision for all shader stages.

<code>D3D12_SHADER_MIN_PRECISION_SUPPORT_10_BIT</code>
--

Value: *0x1*

The driver supports 10-bit precision.

<code>D3D12_SHADER_MIN_PRECISION_SUPPORT_16_BIT</code>
--

Value: *0x2*

The driver supports 16-bit precision.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

The returned info just indicates that the graphics hardware can perform HLSL operations at a lower precision than the standard 32-bit float precision, but doesn't guarantee that the graphics hardware will actually run at a lower precision.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_RESOURCE_VIEW_DESC structure (d3d12.h)

Article 07/27/2022

Describes a shader-resource view (SRV).

Syntax

C++

```
typedef struct D3D12_SHADER_RESOURCE_VIEW_DESC {
    DXGI_FORMAT Format;
    D3D12_SRV_DIMENSION ViewDimension;
    UINT Shader4ComponentMapping;
    union {
        D3D12_BUFFER_SRV Buffer;
        D3D12_TEX1D_SRV Texture1D;
        D3D12_TEX1D_ARRAY_SRV Texture1DArray;
        D3D12_TEX2D_SRV Texture2D;
        D3D12_TEX2D_ARRAY_SRV Texture2DArray;
        D3D12_TEX2DMS_SRV Texture2DMS;
        D3D12_TEX2DMS_ARRAY_SRV Texture2DMSArray;
        D3D12_TEX3D_SRV Texture3D;
        D3D12_TEXCUBE_SRV TextureCube;
        D3D12_TEXCUBE_ARRAY_SRV TextureCubeArray;
        D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV
        RaytracingAccelerationStructure;
    };
} D3D12_SHADER_RESOURCE_VIEW_DESC;
```

Members

Format

A [DXGI_FORMAT](#)-typed value that specifies the viewing format. See remarks.

ViewDimension

A [D3D12_SRV_DIMENSION](#)-typed value that specifies the resource type of the view. This type is the same as the resource type of the underlying resource. This member also determines which _SRV to use in the union below.

Shader4ComponentMapping

A value, constructed using the [D3D12_ENCODE_SHADER_4_COMPONENT_MAPPING](#) macro. The **D3D12_SHADER_COMPONENT_MAPPING** enumeration specifies what values from memory should be returned when the texture is accessed in a shader via this shader resource view (SRV). For example, it can route component 1 (green) from memory, or the constant `0`, into component 2 (`.b`) of the value given to the shader.

Buffer

A [D3D12_BUFFER_SRV](#) structure that views the resource as a buffer.

Texture1D

A [D3D12_TEX1D_SRV](#) structure that views the resource as a 1D texture.

Texture1DArray

A [D3D12_TEX1D_ARRAY_SRV](#) structure that views the resource as a 1D-texture array.

Texture2D

A [D3D12_TEX2D_SRV](#) structure that views the resource as a 2D-texture.

Texture2DArray

A [D3D12_TEX2D_ARRAY_SRV](#) structure that views the resource as a 2D-texture array.

Texture2DMS

A [D3D12_TEX2DMS_SRV](#) structure that views the resource as a 2D-multisampled texture.

Texture2DMSArray

A [D3D12_TEX2DMS_ARRAY_SRV](#) structure that views the resource as a 2D-multisampled-texture array.

Texture3D

A [D3D12_TEX3D_SRV](#) structure that views the resource as a 3D texture.

TextureCube

A [D3D12_TEXCUBE_SRV](#) structure that views the resource as a 3D-cube texture.

TextureCubeArray

A [D3D12_TEXCUBE_ARRAY_SRV](#) structure that views the resource as a 3D-cube-texture array.

RaytracingAccelerationStructure

A [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV](#) structure that views the resource as a raytracing acceleration structure.

Remarks

A view is a format-specific way to look at the data in a resource. The view determines what data to look at, and how it is cast when read.

When viewing a resource, the resource-view description must specify a typed format, that is compatible with the resource format. So that means that you can't create a resource-view description using any format with _TYPELESS in the name. You can however view a typeless resource by specifying a typed format for the view. For example, a DXGI_FORMAT_R32G32B32_TYPELESS resource can be viewed with one of these typed formats: DXGI_FORMAT_R32G32B32_FLOAT, DXGI_FORMAT_R32G32B32_UINT, and DXGI_FORMAT_R32G32B32_SINT, since these typed formats are compatible with the typeless resource.

Create a shader-resource-view description by calling [ID3D12Device::CreateShaderResourceView](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_VISIBILITY enumeration (d3d12.h)

Article02/22/2024

Specifies the shaders that can access the contents of a given root signature slot.

Syntax

C++

```
typedef enum D3D12_SHADER_VISIBILITY {
    D3D12_SHADER_VISIBILITY_ALL = 0,
    D3D12_SHADER_VISIBILITY_VERTEX = 1,
    D3D12_SHADER_VISIBILITY_HULL = 2,
    D3D12_SHADER_VISIBILITY_DOMAIN = 3,
    D3D12_SHADER_VISIBILITY_GEOMETRY = 4,
    D3D12_SHADER_VISIBILITY_PIXEL = 5,
    D3D12_SHADER_VISIBILITY_AMPLIFICATION = 6,
    D3D12_SHADER_VISIBILITY_MESH = 7
} ;
```

Constants

[+] Expand table

D3D12_SHADER_VISIBILITY_ALL Value: 0 Specifies that all shader stages can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_VERTEX Value: 1 Specifies that the vertex shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_HULL Value: 2 Specifies that the hull shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_DOMAIN Value: 3 Specifies that the domain shader stage can access whatever is bound at the root signature slot.

D3D12_SHADER_VISIBILITY_GEOMETRY

Value: 4

Specifies that the geometry shader stage can access whatever is bound at the root signature slot.

D3D12_SHADER_VISIBILITY_PIXEL

Value: 5

Specifies that the pixel shader stage can access whatever is bound at the root signature slot.

D3D12_SHADER_VISIBILITY_AMPLIFICATION

Value: 6

Specifies that the amplification shader stage can access whatever is bound at the root signature slot.

D3D12_SHADER_VISIBILITY_MESH

Value: 7

Specifies that the mesh shader stage can access whatever is bound at the root signature slot.

Remarks

This enum is used by the [D3D12_ROOT_PARAMETER](#) structure.

The compute queue always uses **D3D12_SHADER_VISIBILITY_ALL** because it has only one active stage. The 3D queue can choose values, but if it uses **D3D12_SHADER_VISIBILITY_ALL**, all shader stages can access whatever is bound at the root signature slot.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_SHADING_RATE enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify the shading rate (for variable-rate shading, or VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

C++

```
typedef enum D3D12_SHADING_RATE {
    D3D12_SHADING_RATE_1X1 = 0,
    D3D12_SHADING_RATE_1X2 = 0x1,
    D3D12_SHADING_RATE_2X1 = 0x4,
    D3D12_SHADING_RATE_2X2 = 0x5,
    D3D12_SHADING_RATE_2X4 = 0x6,
    D3D12_SHADING_RATE_4X2 = 0x9,
    D3D12_SHADING_RATE_4X4 = 0xa
} ;
```

Constants

[] [Expand table](#)

`D3D12_SHADING_RATE_1X1`

Value: `0`

Specifies no change to the shading rate.

`D3D12_SHADING_RATE_1X2`

Value: `0x1`

Specifies that the shading rate should reduce vertical resolution 2x.

`D3D12_SHADING_RATE_2X1`

Value: `0x4`

Specifies that the shading rate should reduce horizontal resolution 2x.

`D3D12_SHADING_RATE_2X2`

Value: `0x5`

Specifies that the shading rate should reduce the resolution of both axes 2x.

D3D12_SHADING_RATE_2X4

Value: 0x6

Specifies that the shading rate should reduce horizontal resolution 2x, and reduce vertical resolution 4x.

D3D12_SHADING_RATE_4X2

Value: 0x9

Specifies that the shading rate should reduce horizontal resolution 4x, and reduce vertical resolution 2x.

D3D12_SHADING_RATE_4X4

Value: 0xa

Specifies that the shading rate should reduce the resolution of both axes 4x.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADING_RATE_COMBINER enumeration (d3d12.h)

Article 02/22/2024

Defines constants that specify a shading rate combiner (for variable-rate shading, or VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

C++

```
typedef enum D3D12_SHADING_RATE_COMBINER {
    D3D12_SHADING_RATE_COMBINER_PASSTHROUGH = 0,
    D3D12_SHADING_RATE_COMBINER_OVERRIDE = 1,
    D3D12_SHADING_RATE_COMBINER_MIN = 2,
    D3D12_SHADING_RATE_COMBINER_MAX = 3,
    D3D12_SHADING_RATE_COMBINER_SUM = 4
};
```

Constants

[] Expand table

<code>D3D12_SHADING_RATE_COMBINER_PASSTHROUGH</code>	Value: 0 Specifies the combiner <code>C.xy = A.xy</code> , for combiner (C) and inputs (A and B).
<code>D3D12_SHADING_RATE_COMBINER_OVERRIDE</code>	Value: 1 Specifies the combiner <code>C.xy = B.xy</code> , for combiner (C) and inputs (A and B).
<code>D3D12_SHADING_RATE_COMBINER_MIN</code>	Value: 2 Specifies the combiner <code>C.xy = max(A.xy, B.xy)</code> , for combiner (C) and inputs (A and B).
<code>D3D12_SHADING_RATE_COMBINER_MAX</code>	Value: 3 Specifies the combiner <code>C.xy = min(A.xy, B.xy)</code> , for combiner (C) and inputs (A and B).
<code>D3D12_SHADING_RATE_COMBINER_SUM</code>	Value: 4

Specifies the combiner $C.xy = \min(\maxRate, A.xy + B.xy)$, for combiner (C) and inputs (A and B).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a cross-API sharing support tier.

The resource data formats mentioned are members of the [DXGI_FORMAT](#) enumeration.

Syntax

C++

```
typedef enum D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER {
    D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_0 = 0,
    D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_1,
    D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_2
};
```

Constants

[] Expand table

`D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_0`

Value: 0

Related to [D3D11_SHARED_RESOURCE_TIER::D3D11_SHARED_RESOURCE_TIER_1](#).

Specifies that the most basic level of cross-API sharing is supported, including the following resource data formats.

- * DXGI_FORMAT_R8G8B8A8_UNORM
- * DXGI_FORMAT_R8G8B8A8_UNORM_SRGB
- * DXGI_FORMAT_B8G8R8A8_UNORM
- * DXGI_FORMAT_B8G8R8A8_UNORM_SRGB
- * DXGI_FORMAT_B8G8R8X8_UNORM
- * DXGI_FORMAT_B8G8R8X8_UNORM_SRGB
- * DXGI_FORMAT_R10G10B10A2_UNORM
- * DXGI_FORMAT_R16G16B16A16_FLOAT

`D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_1`

Related to [D3D11_SHARED_RESOURCE_TIER::D3D11_SHARED_RESOURCE_TIER_2](#).

Specifies that cross-API sharing functionality of **D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_0** is supported, plus the following formats.

- * DXGI_FORMAT_R16G16B16A16_TYPELESS
- * DXGI_FORMAT_R10G10B10A2_TYPELESS
- * DXGI_FORMAT_R8G8B8A8_TYPELESS
- * DXGI_FORMAT_R8G8B8X8_TYPELESS
- * DXGI_FORMAT_R16G16_TYPELESS
- * DXGI_FORMAT_R8G8_TYPELESS
- * DXGI_FORMAT_R32_TYPELESS
- * DXGI_FORMAT_R16_TYPELESS
- * DXGI_FORMAT_R8_TYPELESS

This level support is built into WDDM 2.4.

Also see [Extended support for shared Texture2D resources](#).

D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_2

Related to [D3D11_SHARED_RESOURCE_TIER::D3D11_SHARED_RESOURCE_TIER_3](#).

Specifies that cross-API sharing functionality of **D3D12_SHARED_RESOURCE_COMPATIBILITY_TIER_1** is supported, plus the following formats.

- * DXGI_FORMAT_NV12 (also see [Extended NV12 texture support](#))

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SO_DECLARATION_ENTRY structure (d3d12.h)

Article02/22/2024

Describes a vertex element in a vertex buffer in an output slot.

Syntax

C++

```
typedef struct D3D12_SO_DECLARATION_ENTRY {
    UINT      Stream;
    LPCSTR   SemanticName;
    UINT      SemanticIndex;
    BYTE     StartComponent;
    BYTE     ComponentCount;
    BYTE     OutputSlot;
} D3D12_SO_DECLARATION_ENTRY;
```

Members

Stream

Zero-based, stream number.

SemanticName

Type of output element; possible values include: "POSITION", "NORMAL", or "TEXCOORD0". Note that if **SemanticName** is **NULL** then **ComponentCount** can be greater than 4 and the described entry will be a gap in the stream out where no data will be written.

SemanticIndex

Output element's zero-based index. Use, for example, if you have more than one texture coordinate stored in each vertex.

StartComponent

The component of the entry to begin writing out to. Valid values are 0 to 3. For example, if you only wish to output to the y and z components of a position, **StartComponent** is 1 and **ComponentCount** is 2.

ComponentCount

The number of components of the entry to write out to. Valid values are 1 to 4. For example, if you only wish to output to the y and z components of a position, **StartComponent** is 1 and **ComponentCount** is 2. Note that if **SemanticName** is **NULL** then **ComponentCount** can be greater than 4 and the described entry will be a gap in the stream out where no data will be written.

OutputSlot

The associated stream output buffer that is bound to the pipeline. The valid range for **OutputSlot** is 0 to 3.

Remarks

Specify an array of **D3D12_SO_DECLARATION_ENTRY** structures in the **pSODeclaration** member of a [D3D12_STREAM_OUTPUT_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

D3D12_SRV_DIMENSION enumeration (d3d12.h)

Article 02/22/2024

Identifies the type of resource that will be viewed as a shader resource.

Syntax

C++

```
typedef enum D3D12_SRV_DIMENSION {
    D3D12_SRV_DIMENSION_UNKNOWN = 0,
    D3D12_SRV_DIMENSION_BUFFER = 1,
    D3D12_SRV_DIMENSION_TEXTURE1D = 2,
    D3D12_SRV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D12_SRV_DIMENSION_TEXTURE2D = 4,
    D3D12_SRV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D12_SRV_DIMENSION_TEXTURE2DMS = 6,
    D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY = 7,
    D3D12_SRV_DIMENSION_TEXTURE3D = 8,
    D3D12_SRV_DIMENSION_TEXTURECUBE = 9,
    D3D12_SRV_DIMENSION_TEXTURECUBEARRAY = 10,
    D3D12_SRV_DIMENSION_RAYTRACING_ACCELERATION_STRUCTURE = 11
} ;
```

Constants

[] Expand table

D3D12_SRV_DIMENSION_UNKNOWN

Value: 0

The type is unknown.

D3D12_SRV_DIMENSION_BUFFER

Value: 1

The resource is a buffer.

D3D12_SRV_DIMENSION_TEXTURE1D

Value: 2

The resource is a 1D texture.

`D3D12_SRV_DIMENSION_TEXTURE1DARRAY`

Value: 3

The resource is an array of 1D textures.

`D3D12_SRV_DIMENSION_TEXTURE2D`

Value: 4

The resource is a 2D texture.

`D3D12_SRV_DIMENSION_TEXTURE2DARRAY`

Value: 5

The resource is an array of 2D textures.

`D3D12_SRV_DIMENSION_TEXTURE2DMS`

Value: 6

The resource is a multisampling 2D texture.

`D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY`

Value: 7

The resource is an array of multisampling 2D textures.

`D3D12_SRV_DIMENSION_TEXTURE3D`

Value: 8

The resource is a 3D texture.

`D3D12_SRV_DIMENSION_TEXTURECUBE`

Value: 9

The resource is a cube texture.

`D3D12_SRV_DIMENSION_TEXTURECUBEARRAY`

Value: 10

The resource is an array of cube textures.

`D3D12_SRV_DIMENSION_RAYTRACING_ACCELERATION_STRUCTURE`

Value: 11

The resource is a raytracing acceleration structure.

Remarks

These values are used by a shader-resource-view description,

[D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATE_OBJECT_CONFIG structure (d3d12.h)

Article 02/22/2024

Defines general properties of a state object.

Syntax

C++

```
typedef struct D3D12_STATE_OBJECT_CONFIG {
    D3D12_STATE_OBJECT_FLAGS Flags;
} D3D12_STATE_OBJECT_CONFIG;
```

Members

Flags

A value from the [D3D12_STATE_OBJECT_FLAGS](#) flags enumeration that specifies the requirements for the state object.

Remarks

The presence of this subobject in a state object is optional. If present, all exports in the state object must be associated with the same subobject (or one with a matching definition). This consistency requirement does not apply across existing collections that are included in a larger state object.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATE_OBJECT_DESC structure (d3d12.h)

Article 02/22/2024

Description of a state object. Pass a value of this structure type to [ID3D12Device5::CreateStateObject](#).

Syntax

C++

```
typedef struct D3D12_STATE_OBJECT_DESC {
    D3D12_STATE_OBJECT_TYPE      Type;
    UINT                         NumSubobjects;
    const D3D12_STATE_SUBOBJECT *pSubobjects;
} D3D12_STATE_OBJECT_DESC;
```

Members

Type

The type of the state object.

NumSubobjects

Size of the *pSubobjects* array.

pSubobjects

An array of subobject definitions.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATE_OBJECT_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies constraints for state objects. Use values from this enumeration in the [D3D12_STATE_OBJECT_CONFIG](#) structure.

Syntax

C++

```
typedef enum D3D12_STATE_OBJECT_FLAGS {
    D3D12_STATE_OBJECT_FLAG_NONE = 0,
    D3D12_STATE_OBJECT_FLAG_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITIONS =
        0x1,
    D3D12_STATE_OBJECT_FLAG_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS =
        0x2,
    D3D12_STATE_OBJECT_FLAG_ALLOW_STATE_OBJECT_ADDITIONS
} ;
```

Constants

[+] [Expand table](#)

`D3D12_STATE_OBJECT_FLAG_NONE`

Value: *0*

No state object constraints.

`D3D12_STATE_OBJECT_FLAG_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITIONS`

Value: *0x1*

This flag applies to state objects of type collection only. Otherwise this flag is ignored.

The exports from this collection are allowed to have unresolved references (dependencies) that would have to be resolved (defined) when the collection is included in a containing state object, such as a raytracing pipeline state object (RTPSO). This includes depending on externally defined subobject associations to associate an external subobject (e.g. root signature) to a local export.

In the absence of this flag, all exports in this collection must have their dependencies fully locally resolved, including any necessary subobject associations being defined locally. Advanced implementations/drivers will have enough information to compile the code in the collection and not need to keep around any uncompiled code (unless the

`D3D12_STATE_OBJECT_FLAG_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS` flag is set), so that when the collection is used in a containing state object (e.g. RTPSO), minimal work needs to be done by the driver, ideally a “cheap” link at most.

`D3D12_STATE_OBJECT_FLAG_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS`

Value: `0x2`

This flag applies to state objects of type collection only. Otherwise this flag is ignored.

If this collection is included in another state object (e.g. RTPSO), shaders / functions in the rest of the containing state object are allowed to depend on (e.g. call) exports from this collection.

In the absence of this flag (default), exports from this collection cannot be directly referenced by other parts of containing state objects (e.g. RTPSO). This can reduce memory footprint for the collection slightly since drivers don’t need to keep uncompiled code in the collection on the off chance that it may get called by some external function that would then compile all the code together. That said, if not all necessary subobject associations have been locally defined for code in this collection, the driver may not be able to compile shader code yet and may still need to keep uncompiled code around.

A subobject association defined externally that associates an external subobject to a local export does not count as an external dependency on a local definition, so the presence or absence of this flag does not affect whether the association is allowed or not. On the other hand if the current collection defines a subobject association for a locally defined subobject to an external export (e.g. shader), that counts as an external dependency on a local definition and this flag must be set.

Regardless of the presence or absence of this flag, shader entrypoints (such as hit groups or miss shaders) in the collection are visible as entrypoints to a containing state object (e.g. RTPSO) if exported by it. In the case of an RTPSO, the exported entrypoints can be used in shader tables for raytracing.

Requirements

[+] Expand table

Requirement	Value
Header	<code>d3d12.h</code>

Feedback

Was this page helpful?

 Yes

 No

D3D12_STATE_OBJECT_TYPE enumeration (d3d12.h)

Article 02/22/2024

Specifies the type of a state object. Use with [D3D12_STATE_OBJECT_DESC](#).

Syntax

C++

```
typedef enum D3D12_STATE_OBJECT_TYPE {
    D3D12_STATE_OBJECT_TYPE_COLLECTION = 0,
    D3D12_STATE_OBJECT_TYPE_RAYTRACING_PIPELINE = 3,
    D3D12_STATE_OBJECT_TYPE_EXECUTABLE
} ;
```

Constants

[+] Expand table

	<code>D3D12_STATE_OBJECT_TYPE_COLLECTION</code>
Value:	0
Collection state object.	
	<code>D3D12_STATE_OBJECT_TYPE_RAYTRACING_PIPELINE</code>
Value:	3
Raytracing pipeline state object.	

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATE_SUBOBJECT structure (d3d12.h)

Article 02/22/2024

Represents a subobject within a state object description. Use with [D3D12_STATE_OBJECT_DESC](#).

Syntax

C++

```
typedef struct D3D12_STATE_SUBOBJECT {
    D3D12_STATE_SUBOBJECT_TYPE Type;
    const void               *pDesc;
} D3D12_STATE_SUBOBJECT;
```

Members

Type

A [D3D12_STATE_SUBOBJECT_TYPE](#) specifying the type of the state subobject.

pDesc

Pointer to state object description of the type specified in the *Type* parameter.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

Yes

No

D3D12_STATE_SUBOBJECT_TYPE enumeration (d3d12.h)

Article01/31/2022

The type of a state subobject. Use with [D3D12_STATE_SUBOBJECT](#).

Syntax

C++

```
typedef enum D3D12_STATE_SUBOBJECT_TYPE {
    D3D12_STATE_SUBOBJECT_TYPE_STATE_OBJECT_CONFIG = 0,
    D3D12_STATE_SUBOBJECT_TYPE_GLOBAL_ROOT_SIGNATURE = 1,
    D3D12_STATE_SUBOBJECT_TYPE_LOCAL_ROOT_SIGNATURE = 2,
    D3D12_STATE_SUBOBJECT_TYPE_NODE_MASK = 3,
    D3D12_STATE_SUBOBJECT_TYPE_DXIL_LIBRARY = 5,
    D3D12_STATE_SUBOBJECT_TYPE_EXISTING_COLLECTION = 6,
    D3D12_STATE_SUBOBJECT_TYPE_SUBOBJECT_TO_EXPORTS_ASSOCIATION = 7,
    D3D12_STATE_SUBOBJECT_TYPE_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION = 8,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_SHADER_CONFIG = 9,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG = 10,
    D3D12_STATE_SUBOBJECT_TYPE_HIT_GROUP = 11,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG1,
    D3D12_STATE_SUBOBJECT_TYPE_WORK_GRAPH,
    D3D12_STATE_SUBOBJECT_TYPE_STREAM_OUTPUT,
    D3D12_STATE_SUBOBJECT_TYPE_BLEND,
    D3D12_STATE_SUBOBJECT_TYPE_SAMPLE_MASK,
    D3D12_STATE_SUBOBJECT_TYPE_RASTERIZER,
    D3D12_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL,
    D3D12_STATE_SUBOBJECT_TYPE_INPUT_LAYOUT,
    D3D12_STATE_SUBOBJECT_TYPE_IB_STRIP_CUT_VALUE,
    D3D12_STATE_SUBOBJECT_TYPE_PRIMITIVE_TOPOLOGY,
    D3D12_STATE_SUBOBJECT_TYPE_RENDER_TARGET_FORMATS,
    D3D12_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL_FORMAT,
    D3D12_STATE_SUBOBJECT_TYPE_SAMPLE_DESC,
    D3D12_STATE_SUBOBJECT_TYPE_FLAGS,
    D3D12_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL1,
    D3D12_STATE_SUBOBJECT_TYPE_VIEW_INSTANCING,
    D3D12_STATE_SUBOBJECT_TYPE_GENERIC_PROGRAM,
    D3D12_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL2,
    D3D12_STATE_SUBOBJECT_TYPE_MAX_VALID
};
```

Constants

D3D12_STATE_SUBOBJECT_TYPE_STATE_OBJECT_CONFIG

Value: 0

Subobject type is [D3D12_STATE_OBJECT_CONFIG](#).

D3D12_STATE_SUBOBJECT_TYPE_GLOBAL_ROOT_SIGNATURE

Value: 1

Subobject type is [D3D12_GLOBAL_ROOT_SIGNATURE](#).

D3D12_STATE_SUBOBJECT_TYPE_LOCAL_ROOT_SIGNATURE

Value: 2

Subobject type is [D3D12_LOCAL_ROOT_SIGNATURE](#).

D3D12_STATE_SUBOBJECT_TYPE_NODE_MASK

Value: 3

Subobject type is [D3D12_NODE_MASK](#).

Important

On some versions of the DirectX Runtime, specifying a node via

[D3D12_NODE_MASK](#) in a [D3D12_STATE_SUBOBJECT](#) with type

D3D12_STATE_SUBOBJECT_TYPE_NODE_MASK, the runtime will incorrectly handle a node mask value of 0, which should use node #1, which will lead to errors when attempting to use the state object later. Specify an explicit node value of 1, or omit the [D3D12_NODE_MASK](#) subobject to avoid this issue.

D3D12_STATE_SUBOBJECT_TYPE_DXIL_LIBRARY

Value: 5

Subobject type is [D3D12_DXIL_LIBRARY_DESC](#).

D3D12_STATE_SUBOBJECT_TYPE_EXISTING_COLLECTION

Value: 6

Subobject type is [D3D12_EXISTING_COLLECTION_DESC](#).

D3D12_STATE_SUBOBJECT_TYPE_SUBOBJECT_TO_EXPORTS_ASSOCIATION

Value: 7

Subobject type is [D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION](#).

D3D12_STATE_SUBOBJECT_TYPE_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION

Value: 8

Subobject type is [D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION](#).

D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_SHADER_CONFIG

Value: 9

Subobject type is [D3D12_RAYTRACING_SHADER_CONFIG](#).

[D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG](#)

Value: 10

Subobject type is [D3D12_RAYTRACING_PIPELINE_CONFIG](#).

[D3D12_STATE_SUBOBJECT_TYPE_HIT_GROUP](#)

Value: 11

Subobject type is [D3D12_HIT_GROUP_DESC](#)

[D3D12_STATE_SUBOBJECT_TYPE_MAX_VALID](#)

The maximum valid subobject type value.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATIC_BORDER_COLOR enumeration (d3d12.h)

Article02/22/2024

Specifies the border color for a static sampler.

Syntax

C++

```
typedef enum D3D12_STATIC_BORDER_COLOR {
    D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK = 0,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_WHITE,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK_UINT,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_WHITE_UINT
} ;
```

Constants

[+] Expand table

D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK

Value: 0

Indicates black, with the alpha component as fully transparent.

D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK
--

Indicates black, with the alpha component as fully opaque.

D3D12_STATIC_BORDER_COLOR_OPAQUE_WHITE
--

Indicates white, with the alpha component as fully opaque.

Remarks

This enum is used by the [D3D12_STATIC_SAMPLER_DESC](#) structure.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STATIC_SAMPLER_DESC structure (d3d12.h)

Article 10/06/2021

Describes a static sampler.

Syntax

C++

```
typedef struct D3D12_STATIC_SAMPLER_DESC {
    D3D12_FILTER             Filter;
    D3D12_TEXTURE_ADDRESS_MODE AddressU;
    D3D12_TEXTURE_ADDRESS_MODE AddressV;
    D3D12_TEXTURE_ADDRESS_MODE AddressW;
    FLOAT                     MipLODBias;
    UINT                      MaxAnisotropy;
    D3D12_COMPARISON_FUNC     ComparisonFunc;
    D3D12_STATIC_BORDER_COLOR BorderColor;
    FLOAT                     MinLOD;
    FLOAT                     MaxLOD;
    UINT                      ShaderRegister;
    UINT                      RegisterSpace;
    D3D12_SHADER_VISIBILITY   ShaderVisibility;
} D3D12_STATIC_SAMPLER_DESC;
```

Members

Filter

The filtering method to use when sampling a texture, as a [D3D12_FILTER](#) enumeration constant.

AddressU

Specifies the [D3D12_TEXTURE_ADDRESS_MODE](#) mode to use for resolving a *u* texture coordinate that is outside the 0 to 1 range.

AddressV

Specifies the [D3D12_TEXTURE_ADDRESS_MODE](#) mode to use for resolving a *v* texture coordinate that is outside the 0 to 1 range.

AddressW

Specifies the [D3D12_TEXTURE_ADDRESS_MODE](#) mode to use for resolving a w texture coordinate that is outside the 0 to 1 range.

MipLODBias

Offset from the calculated mipmap level. For example, if Direct3D calculates that a texture should be sampled at mipmap level 3 and MipLODBias is 2, then the texture will be sampled at mipmap level 5.

MaxAnisotropy

Clamping value used if D3D12_FILTER_ANISOTROPIC or D3D12_FILTER_COMPARISON_ANISOTROPIC is specified as the filter. Valid values are between 1 and 16.

ComparisonFunc

A function that compares sampled data against existing sampled data. The function options are listed in [D3D12_COMPARISON_FUNC](#).

BorderColor

One member of [D3D12_STATIC_BORDER_COLOR](#), the border color to use if D3D12_TEXTURE_ADDRESS_MODE_BORDER is specified for AddressU, AddressV, or AddressW. Range must be between 0.0 and 1.0 inclusive.

MinLOD

Lower end of the mipmap range to clamp access to, where 0 is the largest and most detailed mipmap level and any level higher than that is less detailed.

MaxLOD

Upper end of the mipmap range to clamp access to, where 0 is the largest and most detailed mipmap level and any level higher than that is less detailed. This value must be greater than or equal to MinLOD. To have no upper limit on LOD set this to a large value such as D3D12_FLOAT32_MAX.

ShaderRegister

The *ShaderRegister* and *RegisterSpace* parameters correspond to the binding syntax of HLSL. For example, in HLSL:

Syntax

```
Texture2D<float4> a : register(t2, space3);
```

This corresponds to a *ShaderRegister* of 2 (indicating the type is SRV), and *RegisterSpace* is 3.

The *ShaderRegister* and *RegisterSpace* pair is needed to establish correspondence between shader resources and runtime heap descriptors, using the root signature data structure.

RegisterSpace

See the description for *ShaderRegister*. Register space is optional; the default register space is 0.

ShaderVisibility

Specifies the visibility of the sampler to the pipeline shaders, one member of [D3D12_SHADER_VISIBILITY](#).

Remarks

Use this structure with the [D3D12_ROOT_SIGNATURE_DESC](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_STATIC_SAMPLER_DESC](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_STENCIL_OP enumeration (d3d12.h)

Article01/31/2022

Identifies the stencil operations that can be performed during depth-stencil testing.

Syntax

C++

```
typedef enum D3D12_STENCIL_OP {
    D3D12_STENCIL_OP_KEEP = 1,
    D3D12_STENCIL_OP_ZERO = 2,
    D3D12_STENCIL_OP_REPLACE = 3,
    D3D12_STENCIL_OP_INCR_SAT = 4,
    D3D12_STENCIL_OP_DECR_SAT = 5,
    D3D12_STENCIL_OP_INVERT = 6,
    D3D12_STENCIL_OP_INCR = 7,
    D3D12_STENCIL_OP_DECR = 8
} ;
```

Constants

[+] Expand table

D3D12_STENCIL_OP_KEEP Value: 1 Keep the existing stencil data.
D3D12_STENCIL_OP_ZERO Value: 2 Set the stencil data to 0.
D3D12_STENCIL_OP_REPLACE Value: 3 Set the stencil data to the reference value set by calling ID3D12GraphicsCommandList::OMSetStencilRef .
D3D12_STENCIL_OP_INCR_SAT Value: 4 Increment the stencil value by 1, and clamp the result.

D3D12_STENCIL_OP_DECL_SAT

Value: 5

Decrement the stencil value by 1, and clamp the result.

D3D12_STENCIL_OP_INVERT

Value: 6

Invert the stencil data.

D3D12_STENCIL_OP_INCR

Value: 7

Increment the stencil value by 1, and wrap the result if necessary.

D3D12_STENCIL_OP_DECR

Value: 8

Decrement the stencil value by 1, and wrap the result if necessary.

Remarks

This enum is used by the [D3D12_DEPTH_STENCILOP_DESC](#) structure.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STREAM_OUTPUT_BUFFER_VIEW structure (d3d12.h)

Article04/02/2021

Describes a stream output buffer.

Syntax

C++

```
typedef struct D3D12_STREAM_OUTPUT_BUFFER_VIEW {
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation;
    UINT64                  SizeInBytes;
    D3D12_GPU_VIRTUAL_ADDRESS BufferFilledSizeLocation;
} D3D12_STREAM_OUTPUT_BUFFER_VIEW;
```

Members

BufferLocation

A D3D12_GPU_VIRTUAL_ADDRESS (a **UINT64**) that points to the stream output buffer. If **SizeInBytes** is 0, this member isn't used and can be any value.

SizeInBytes

The size of the stream output buffer in bytes.

BufferFilledSizeLocation

The location of the value of how much data has been filled into the buffer, as a D3D12_GPU_VIRTUAL_ADDRESS (a **UINT64**). This member can't be NULL; a filled size location must be supplied (which the hardware will increment as data is output). If **SizeInBytes** is 0, this member isn't used and can be any value.

Remarks

Use this structure with [SOSetTargets](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_STREAM_OUTPUT_DESC structure (d3d12.h)

Article 02/22/2024

Describes a streaming output buffer.

Syntax

C++

```
typedef struct D3D12_STREAM_OUTPUT_DESC {
    const D3D12_SO_DECLARATION_ENTRY *pSODeclaration;
    UINT NumEntries;
    const UINT *pBufferStrides;
    UINT NumStrides;
    UINT RasterizedStream;
} D3D12_STREAM_OUTPUT_DESC;
```

Members

pSODeclaration

An array of [D3D12_SO_DECLARATION_ENTRY](#) structures. Can't be NULL if **NumEntries** > 0.

NumEntries

The number of entries in the stream output declaration array that the **pSODeclaration** member points to.

pBufferStrides

An array of buffer strides; each stride is the size of an element for that buffer.

NumStrides

The number of strides (or buffers) that the **pBufferStrides** member points to.

RasterizedStream

The index number of the stream to be sent to the rasterizer stage.

Remarks

A [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) object contains a [D3D12_STREAM_OUTPUT_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure (d3d12.h)

Article 02/22/2024

Associates a subobject defined directly in a state object with shader exports.

Syntax

C++

```
typedef struct D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION {
    const D3D12_STATE_SUBOBJECT *pSubobjectToAssociate;
    UINT                         NumExports;
    LPCWSTR                       *pExports;
} D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION;
```

Members

pSubobjectToAssociate

Pointer to the subobject in current state object to define an association to.

NumExports

Size of the *pExports* array. If 0, this is being explicitly defined as a default association.

Another way to define a default association is to omit this subobject association for that subobject completely.

pExports

The array of exports with which the subobject is associated.

Remarks

Depending on the flags specified in the optional [D3D12_STATE_OBJECT_CONFIG](#) subobject for opting into cross linkage, the exports being associated don't necessarily have to be present in the current state object, or one that has been seen yet, to be resolved later, on raytracing pipeline state object (RTPSO) creation for example.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_SUBRESOURCE_DATA structure (d3d12.h)

Article 02/22/2024

Describes subresource data.

Syntax

C++

```
typedef struct D3D12_SUBRESOURCE_DATA {
    const void *pData;
    LONG_PTR RowPitch;
    LONG_PTR SlicePitch;
} D3D12_SUBRESOURCE_DATA;
```

Members

`pData`

A pointer to a memory block that contains the subresource data.

`RowPitch`

The row pitch, or width, or physical size, in bytes, of the subresource data.

`SlicePitch`

The depth pitch, or width, or physical size, in bytes, of the subresource data.

Remarks

This structure is used by a number of the helper functions, refer to [Helper Structures and Functions for D3D12](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SUBRESOURCE_FOOTPRINT structure (d3d12.h)

Article02/22/2024

Describes the format, width, height, depth, and row-pitch of the subresource into the parent resource.

Syntax

C++

```
typedef struct D3D12_SUBRESOURCE_FOOTPRINT {
    DXGI_FORMAT Format;
    UINT Width;
    UINT Height;
    UINT Depth;
    UINT RowPitch;
} D3D12_SUBRESOURCE_FOOTPRINT;
```

Members

Format

A [DXGI_FORMAT](#)-typed value that specifies the viewing format.

Width

The width of the subresource.

Height

The height of the subresource.

Depth

The depth of the subresource.

RowPitch

The row pitch, or width, or physical size, in bytes, of the subresource data. This must be a multiple of [D3D12_TEXTURE_DATA_PITCH_ALIGNMENT](#) (256), and must be greater than or equal to the size of the data within a row.

Remarks

Use this structure in the [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#) structure.

The helper structure is [CD3DX12_SUBRESOURCE_FOOTPRINT](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_SUBRESOURCE_FOOTPRINT](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SUBRESOURCE_INFO structure (d3d12.h)

Article 02/22/2024

Describes subresource data.

Syntax

C++

```
typedef struct D3D12_SUBRESOURCE_INFO {
    UINT64 Offset;
    UINT    RowPitch;
    UINT    DepthPitch;
} D3D12_SUBRESOURCE_INFO;
```

Members

Offset

Offset, in bytes, between the start of the parent resource and this subresource.

RowPitch

The row pitch, or width, or physical size, in bytes, of the subresource data.

DepthPitch

The depth pitch, or width, or physical size, in bytes, of the subresource data.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SUBRESOURCE_RANGE_UINT64 structure (d3d12.h)

Article 02/22/2024

Describes a subresource memory range.

Syntax

C++

```
typedef struct D3D12_SUBRESOURCE_RANGE_UINT64 {
    UINT             Subresource;
    D3D12_RANGE_UINT64 Range;
} D3D12_SUBRESOURCE_RANGE_UINT64;
```

Members

Subresource

The index of the subresource.

Range

A memory range within the subresource.

Remarks

This structure is used by the [AtomicCopyBufferUINT](#) and [AtomicCopyBufferUINT64](#) methods.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SUBRESOURCE_TILING structure (d3d12.h)

Article 02/22/2024

Describes a tiled subresource volume.

Syntax

C++

```
typedef struct D3D12_SUBRESOURCE_TILING {
    UINT    WidthInTiles;
    UINT16 HeightInTiles;
    UINT16 DepthInTiles;
    UINT    StartTileIndexInOverallResource;
} D3D12_SUBRESOURCE_TILING;
```

Members

`WidthInTiles`

The width in tiles of the subresource.

`HeightInTiles`

The height in tiles of the subresource.

`DepthInTiles`

The depth in tiles of the subresource.

`StartTileIndexInOverallResource`

The index of the tile in the overall tiled subresource to start with.

Remarks

This structure is used by the [GetResourceTiling](#) method.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_SUBRESOURCE_TILING](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_ARRAY_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of 1D textures to use in a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX1D_ARRAY_DSV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX1D_ARRAY_DSV;
```

Members

MipSlice

The index of the first mipmap level to use.

FirstArraySlice

The index of the first texture to use in an array of textures.

ArraySize

Number of textures to use.

Remarks

Use this structure with a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure to view the resource as an array of 1D textures.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_ARRAY_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of 1D textures to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX1D_ARRAY_RTV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX1D_ARRAY_RTV;
```

Members

MipSlice

The index of the mipmap level to use mip slice.

FirstArraySlice

The index of the first texture to use in an array of textures.

ArraySize

Number of textures to use.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as an array of 1D textures.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_ARRAY_SRV structure (d3d12.h)

Article11/19/2022

Describes the subresources from an array of 1D textures to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX1D_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT FirstArraySlice;
    UINT ArraySize;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX1D_ARRAY_SRV;
```

Members

`MostDetailedMip`

Index of the most detailed mipmap level to use; this number is between 0 and `MipLevels` (from the original Texture1D for which [ID3D12Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#).

Set to -1 to indicate all the mipmap levels from `MostDetailedMip` on down to least detailed.

`FirstArraySlice`

The index of the first texture to use in an array of textures.

`ArraySize`

Number of textures in the array.

ResourceMinLODClamp

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to *MipCount* - 1.

We recommend that you don't set *MostDetailedMip* and *ResourceMinLODClamp* at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_ARRAY_UAV structure (d3d12.h)

Article 02/22/2024

Describes an array of unordered-access 1D texture resources.

Syntax

C++

```
typedef struct D3D12_TEX1D_ARRAY_UAV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX1D_ARRAY_UAV;
```

Members

MipSlice

The mipmap slice index.

FirstArraySlice

The zero-based index of the first array slice to be accessed.

ArraySize

The number of slices in the array.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as an array of 1D textures.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a 1D texture that is accessible to a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX1D_DSV {  
    UINT MipSlice;  
} D3D12_TEX1D_DSV;
```

Members

MipSlice

The index of the first mipmap level to use.

Remarks

Use this structure with a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure to view the resource as a 1D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a 1D texture to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX1D_RTV {  
    UINT MipSlice;  
} D3D12_TEX1D_RTV;
```

Members

MipSlice

The index of the mipmap level to use mip slice.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as a 1D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_SRV structure (d3d12.h)

Article 02/22/2024

Specifies the subresource from a 1D texture to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX1D_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX1D_SRV;
```

Members

MostDetailedMip

Index of the most detailed mipmap level to use; this number is between 0 and **MipLevels** (from the original Texture1D for which [ID3D12Device::CreateShaderResourceView](#) creates a view) -1.

MipLevels

The maximum number of mipmap levels for the view of the texture. See the remarks.

Set to -1 to indicate all the mipmap levels from **MostDetailedMip** on down to least detailed.

ResourceMinLODClamp

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to *MipCount* - 1.

We recommend that you don't set *MostDetailedMip* and *ResourceMinLODClamp* at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].

- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

As an example, assuming **MostDetailedMip** = 6 and **MipLevels** = 2, the view will have access to 2 mipmap levels, 6 and 7, of the original texture for which [ID3D12Device::CreateShaderResourceView](#) creates the view. In this situation, **MostDetailedMip** is greater than the **MipLevels** in the view. However, **MostDetailedMip** is not greater than the **MipLevels** in the original resource.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX1D_UAV structure (d3d12.h)

Article 02/22/2024

Describes a unordered-access 1D texture resource.

Syntax

C++

```
typedef struct D3D12_TEX1D_UAV {
    UINT MipSlice;
} D3D12_TEX1D_UAV;
```

Members

MipSlice

The mipmap slice index.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as a 1D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_ARRAY_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of 2D textures that are accessible to a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX2D_ARRAY_DSV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX2D_ARRAY_DSV;
```

Members

MipSlice

The index of the first mipmap level to use.

FirstArraySlice

The index of the first texture to use in an array of textures.

ArraySize

Number of textures to use.

Remarks

Use this structure with a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure to view the resource as an array of 2D textures.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_ARRAY_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of 2D textures to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX2D_ARRAY_RTV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
} D3D12_TEX2D_ARRAY_RTV;
```

Members

MipSlice

The index of the mipmap level to use mip slice.

FirstArraySlice

The index of the first texture to use in an array of textures.

ArraySize

Number of textures in the array to use in the render target view, starting from **FirstArraySlice**.

PlaneSlice

The index (plane slice number) of the plane to use in an array of textures.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as an array of 2D textures.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_ARRAY_SRV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of 2D textures to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX2D_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX2D_ARRAY_SRV;
```

Members

`MostDetailedMip`

Index of the most detailed mipmap level to use; this number is between 0 and `MipLevels` - 1 (where `MipLevels` is from the original Texture2D for which `ID3D12Device::CreateShaderResourceView` creates a view).

`MipLevels`

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#).

Set to -1 to indicate all the mipmap levels from `MostDetailedMip` on down to least detailed.

`FirstArraySlice`

The index of the first texture to use in an array of textures.

`ArraySize`

Number of textures in the array.

PlaneSlice

The index (plane slice number) of the plane to use in an array of textures.

ResourceMinLODClamp

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to *MipCount* - 1.

We recommend that you don't set *MostDetailedMip* and *ResourceMinLODClamp* at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_TEX2D_ARRAY_UAV structure (d3d12.h)

Article 02/22/2024

Describes an array of unordered-access 2D texture resources.

Syntax

C++

```
typedef struct D3D12_TEX2D_ARRAY_UAV {
    UINT MipSlice;
    UINT FirstArraySlice;
    UINT ArraySize;
    UINT PlaneSlice;
} D3D12_TEX2D_ARRAY_UAV;
```

Members

MipSlice

The mipmap slice index.

FirstArraySlice

The zero-based index of the first array slice to be accessed.

ArraySize

The number of slices in the array.

PlaneSlice

The index (plane slice number) of the plane to use in an array of textures.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as an array of 2D textures.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a 2D texture that is accessible to a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX2D_DSV {  
    UINT MipSlice;  
} D3D12_TEX2D_DSV;
```

Members

MipSlice

The index of the first mipmap level to use.

Remarks

Use this structure with a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure to view the resource as a 2D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a 2D texture to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX2D_RTV {  
    UINT MipSlice;  
    UINT PlaneSlice;  
} D3D12_TEX2D_RTV;
```

Members

MipSlice

The index of the mipmap level to use.

PlaneSlice

The index (plane slice number) of the plane to use in the texture.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as a 2D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_SRV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a 2D texture to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX2D_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT PlaneSlice;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX2D_SRV;
```

Members

`MostDetailedMip`

The index of the most detailed mipmap level to use; this number is between 0 and `MipLevels` (from the original Texture2D for which `ID3D12Device::CreateShaderResourceView` creates a view) -1.

`MipLevels`

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#). Set to -1 to indicate all the mipmap levels from `MostDetailedMip` on down to least detailed.

`PlaneSlice`

The index (plane slice number) of the plane to use in the texture.

`ResourceMinLODClamp`

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to `MipCount` - 1.

We recommend that you don't set `MostDetailedMip` and `ResourceMinLODClamp` at the same time. Instead, set one of those two members to 0 (to get default behavior). That's

because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2D_UAV structure (d3d12.h)

Article 02/22/2024

Describes a unordered-access 2D texture resource.

Syntax

C++

```
typedef struct D3D12_TEX2D_UAV {
    UINT MipSlice;
    UINT PlaneSlice;
} D3D12_TEX2D_UAV;
```

Members

MipSlice

The mipmap slice index.

PlaneSlice

The index (plane slice number) of the plane to use in the texture.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as a 2D texture.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_ARRAY_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of multi sampled 2D textures for a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_ARRAY_DSV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX2DMS_ARRAY_DSV;
```

Members

`FirstArraySlice`

The index of the first texture to use in an array of textures.

`ArraySize`

Number of textures to use.

Remarks

Use this structure with a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure to view the resource as an array of multi sampled 2D textures.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_ARRAY_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of multi sampled 2D textures to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_ARRAY_RTV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX2DMS_ARRAY_RTV;
```

Members

`FirstArraySlice`

The index of the first texture to use in an array of textures.

`ArraySize`

The number of textures to use.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as an array of multi sampled 2D textures.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_ARRAY_SRV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from an array of multi sampled 2D textures to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_ARRAY_SRV {
    UINT FirstArraySlice;
    UINT ArraySize;
} D3D12_TEX2DMS_ARRAY_SRV;
```

Members

`FirstArraySlice`

The index of the first texture to use in an array of textures.

`ArraySize`

Number of textures to use.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_DSV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a multi sampled 2D texture that is accessible to a depth-stencil view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_DSV {
    UINT UnusedField_NothingToDefine;
} D3D12_TEX2DMS_DSV;
```

Members

UnusedField_NothingToDefine

Unused.

Remarks

This structure is a member of the [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure.

Because a multi sampled 2D texture contains a single subresource, there is nothing to specify in **D3D12_TEX2DMS_DSV**. Consequently, **UnusedField_NothingToDefine** is included so that this structure will compile in C.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a multi sampled 2D texture to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_RTV {
    UINT UnusedField_NothingToDefine;
} D3D12_TEX2DMS_RTV;
```

Members

UnusedField_NothingToDefine

Integer of any value. See remarks.

Remarks

This structure is a member of the [D3D12_RENDER_TARGET_VIEW_DESC](#) structure.

Because a multi sampled 2D texture contains a single subresource, there is actually nothing to specify in **D3D12_TEX2DMS_RTV**. Consequently, **UnusedField_NothingToDefine** is included so that this structure will compile in C.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX2DMS_SRV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from a multi sampled 2D texture to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX2DMS_SRV {
    UINT UnusedField_NothingToDefine;
} D3D12_TEX2DMS_SRV;
```

Members

UnusedField_NothingToDefine

Integer of any value. See remarks.

Remarks

This structure is a member of the [D3D12_SHADER_RESOURCE_VIEW_DESC](#) structure.

Since a multi sampled 2D texture contains a single subresource, there is actually nothing to specify in `D3D12_TEX2DMS_SRV`. Consequently, `UnusedField_NothingToDefine` is included so that this structure will compile in C.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX3D_RTV structure (d3d12.h)

Article 02/22/2024

Describes the subresources from a 3D texture to use in a render-target view.

Syntax

C++

```
typedef struct D3D12_TEX3D_RTV {
    UINT MipSlice;
    UINT FirstWSlice;
    UINT WSize;
} D3D12_TEX3D_RTV;
```

Members

MipSlice

The index of the mipmap level to use mip slice.

FirstWSlice

First depth level to use.

WSize

Number of depth levels to use in the render-target view, starting from **FirstWSlice**. A value of -1 indicates all of the slices along the w axis, starting from **FirstWSlice**.

Remarks

Use this structure with a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure to view the resource as a 3D texture.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX3D_SRV structure (d3d12.h)

Article02/22/2024

Describes the subresources from a 3D texture to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEX3D_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    FLOAT ResourceMinLODClamp;
} D3D12_TEX3D_SRV;
```

Members

`MostDetailedMip`

Index of the most detailed mipmap level to use; this number is between 0 and `MipLevels` (from the original Texture3D for which [ID3D12Device::CreateShaderResourceView](#) creates a view) -1.

`MipLevels`

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#).

Set to -1 to indicate all the mipmap levels from `MostDetailedMip` on down to least detailed.

`ResourceMinLODClamp`

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to `MipCount` - 1.

We recommend that you don't set `MostDetailedMip` and `ResourceMinLODClamp` at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because `MipLevels` is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEX3D_UAV structure (d3d12.h)

Article 02/22/2024

Describes a unordered-access 3D texture resource.

Syntax

C++

```
typedef struct D3D12_TEX3D_UAV {
    UINT MipSlice;
    UINT FirstWSlice;
    UINT WSize;
} D3D12_TEX3D_UAV;
```

Members

MipSlice

The mipmap slice index.

FirstWSlice

The zero-based index of the first depth slice to be accessed.

WSize

The number of depth slices.

Set to -1 to indicate all the depth slices from `FirstWSlice` to the last slice.

Remarks

Use this structure with a `D3D12_UNORDERED_ACCESS_VIEW_DESC` structure to view the resource as a 3D texture.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXCUBE_ARRAY_SRV structure (d3d12.h)

Article11/19/2022

Describes the subresources from an array of cube textures to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEXCUBE_ARRAY_SRV {
    UINT MostDetailedMip;
    UINT MipLevels;
    UINT First2DArrayFace;
    UINT NumCubes;
    FLOAT ResourceMinLODClamp;
} D3D12_TEXCUBE_ARRAY_SRV;
```

Members

MostDetailedMip

Index of the most detailed mipmap level to use; this number is between 0 and **MipLevels** (from the original TextureCube for which [ID3D12Device::CreateShaderResourceView](#) creates a view) -1.

MipLevels

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#).

Set to -1 to indicate all the mipmap levels from **MostDetailedMip** on down to least detailed.

First2DArrayFace

Index of the first 2D texture to use.

NumCubes

Number of cube textures in the array.

ResourceMinLODClamp

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to *MipCount* - 1.

We recommend that you don't set *MostDetailedMip* and *ResourceMinLODClamp* at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXCUBE_SRV structure (d3d12.h)

Article 02/22/2024

Describes the subresource from a cube texture to use in a shader-resource view.

Syntax

C++

```
typedef struct D3D12_TEXCUBE_SRV {
    UINT    MostDetailedMip;
    UINT    MipLevels;
    FLOAT   ResourceMinLODClamp;
} D3D12_TEXCUBE_SRV;
```

Members

MostDetailedMip

Index of the most detailed mipmap level to use; this number is between 0 and **MipLevels** (from the original TextureCube for which [ID3D12Device::CreateShaderResourceView](#) creates a view) -1.

MipLevels

The maximum number of mipmap levels for the view of the texture. See the remarks in [D3D12_TEX1D_SRV](#).

Set to -1 to indicate all the mipmap levels from **MostDetailedMip** on down to least detailed.

ResourceMinLODClamp

Specifies the minimum mipmap level that you can access. Specifying 0.0f means that you can access all of the mipmap levels. Specifying 3.0f means that you can access mipmap levels from 3.0f to *MipCount* - 1.

We recommend that you don't set *MostDetailedMip* and *ResourceMinLODClamp* at the same time. Instead, set one of those two members to 0 (to get default behavior). That's because *MipLevels* is interpreted differently in conjunction with different fields:

- For *MostDetailedMip*, mips are in the range [*MostDetailedMip*, *MostDetailedMip* + *MipLevels* - 1].
- For *ResourceMinLODClamp*, mips are in the range [*ResourceMinLODClamp*, *MipLevels* - 1].

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_ADDRESS_MODE enumeration (d3d12.h)

Article01/31/2022

Identifies a technique for resolving texture coordinates that are outside of the boundaries of a texture.

Syntax

C++

```
typedef enum D3D12_TEXTURE_ADDRESS_MODE {
    D3D12_TEXTURE_ADDRESS_MODE_WRAP = 1,
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR = 2,
    D3D12_TEXTURE_ADDRESS_MODE_CLAMP = 3,
    D3D12_TEXTURE_ADDRESS_MODE_BORDER = 4,
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE = 5
};
```

Constants

[] Expand table

`D3D12_TEXTURE_ADDRESS_MODE_WRAP`

Value: 1

Tile the texture at every (u,v) integer junction.

For example, for u values between 0 and 3, the texture is repeated three times.

`D3D12_TEXTURE_ADDRESS_MODE_MIRROR`

Value: 2

Flip the texture at every (u,v) integer junction.

For u values between 0 and 1, for example, the texture is addressed normally; between 1 and 2, the texture is flipped (mirrored); between 2 and 3, the texture is normal again; and so on.

`D3D12_TEXTURE_ADDRESS_MODE_CLAMP`

Value: 3

Texture coordinates outside the range [0.0, 1.0] are set to the texture color at 0.0 or 1.0, respectively.

`D3D12_TEXTURE_ADDRESS_MODE_BORDER`

Value: 4

Texture coordinates outside the range [0.0, 1.0] are set to the border color specified in [D3D12_SAMPLER_DESC](#) or HLSL code.

`D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE`

Value: 5

Similar to

[D3D12_TEXTURE_ADDRESS_MODE_MIRROR](#)

and

[D3D12_TEXTURE_ADDRESS_MODE_CLAMP](#).

Takes the absolute value of the texture coordinate (thus, mirroring around 0), and then clamps to the maximum value.

Remarks

This enum is used by the [D3D12_SAMPLER_DESC](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_BARRIER structure (d3d12.h)

Article 02/22/2024

Describes a texture memory access barrier. Used by texture barriers to indicate when resource memory must be made visible for a specific access type. Layout transitions are needed only for textures.

Syntax

C++

```
typedef struct D3D12_TEXTURE_BARRIER {
    D3D12_BARRIER_SYNC           SyncBefore;
    D3D12_BARRIER_SYNC           SyncAfter;
    D3D12_BARRIER_ACCESS         AccessBefore;
    D3D12_BARRIER_ACCESS         AccessAfter;
    D3D12_BARRIER_LAYOUT         LayoutBefore;
    D3D12_BARRIER_LAYOUT         LayoutAfter;
    ID3D12Resource               *pResource;
    D3D12_BARRIER_SUBRESOURCE_RANGE Subresources;
    D3D12_TEXTURE_BARRIER_FLAGS   Flags;
} D3D12_TEXTURE_BARRIER;
```

Members

SyncBefore

Synchronization scope of all preceding GPU work that must be completed before executing the barrier.

SyncAfter

Synchronization scope of all subsequent GPU work that must wait until the barrier execution is finished.

AccessBefore

Access bits corresponding with resource usage since the preceding barrier or the start of **ExecuteCommandLists** scope.

AccessAfter

Access bits corresponding with resource usage after the barrier completes.

LayoutBefore

Layout of texture preceding the barrier execution.

LayoutAfter

Layout of texture upon completion of barrier execution.

pResource

Pointer to the buffer resource being using the barrier.

Subresources

Range of texture subresources being bariered.

Flags

Optional flags values.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_BARRIER_FLAGS enumeration (d3d12.h)

Article02/22/2024

Syntax

C++

```
typedef enum D3D12_TEXTURE_BARRIER_FLAGS {
    D3D12_TEXTURE_BARRIER_FLAG_NONE,
    D3D12_TEXTURE_BARRIER_FLAG_DISCARD
} ;
```

Constants

[] Expand table

D3D12_TEXTURE_BARRIER_FLAG_NONE
D3D12_TEXTURE_BARRIER_FLAG_DISCARD

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_COPY_LOCATION structure (d3d12.h)

Article02/22/2024

Describes a portion of a texture for the purpose of texture copies.

Syntax

C++

```
typedef struct D3D12_TEXTURE_COPY_LOCATION {
    ID3D12Resource           *pResource;
    D3D12_TEXTURE_COPY_TYPE  Type;
    union {
        D3D12_PLACED_SUBRESOURCE_FOOTPRINT PlacedFootprint;
        UINT                           SubresourceIndex;
    };
} D3D12_TEXTURE_COPY_LOCATION;
```

Members

pResource

Specifies the resource which will be used for the copy operation.

When **Type** is D3D12_TEXTURE_COPY_TYPE_PLACED_FOOTPRINT, **pResource** must point to a buffer resource.

When **Type** is D3D12_TEXTURE_COPY_TYPE_SUBRESOURCE_INDEX, **pResource** must point to a texture resource.

Type

Specifies which type of resource location this is: a subresource of a texture, or a description of a texture layout which can be applied to a buffer. This [D3D12_TEXTURE_COPY_TYPE](#) enum indicates which union member to use.

PlacedFootprint

Specifies a texture layout, with offset, dimensions, and pitches, for the hardware to understand how to treat a section of a buffer resource as a multi-dimensional texture. To

fill-in the correct data for a [CopyTextureRegion](#) call, see [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#).

SubresourceIndex

Specifies the index of the subresource of an arrayed, mip-mapped, or planar texture should be used for the copy operation.

Remarks

Use this structure with [CopyTextureRegion](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_TEXTURE_COPY_LOCATION](#)

[Core Structures](#)

[D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_COPY_TYPE enumeration (d3d12.h)

Article02/22/2024

Specifies what type of texture copy is to take place.

Syntax

C++

```
typedef enum D3D12_TEXTURE_COPY_TYPE {
    D3D12_TEXTURE_COPY_TYPE_SUBRESOURCE_INDEX = 0,
    D3D12_TEXTURE_COPY_TYPE_PLACED_FOOTPRINT = 1
};
```

Constants

[+] Expand table

<code>D3D12_TEXTURE_COPY_TYPE_SUBRESOURCE_INDEX</code>
--

Value: 0

Indicates a subresource, identified by an index, is to be copied.

<code>D3D12_TEXTURE_COPY_TYPE_PLACED_FOOTPRINT</code>

Value: 1

Indicates a place footprint, identified by a [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#) structure, is to be copied.

Remarks

This enum is used by the [D3D12_TEXTURE_COPY_LOCATION](#) structure.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TEXTURE_LAYOUT enumeration (d3d12.h)

Article07/27/2022

Specifies texture layout options.

Syntax

C++

```
typedef enum D3D12_TEXTURE_LAYOUT {
    D3D12_TEXTURE_LAYOUT_UNKNOWN = 0,
    D3D12_TEXTURE_LAYOUT_ROW_MAJOR = 1,
    D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE = 2,
    D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE = 3
};
```

Constants

[+] Expand table

D3D12_TEXTURE_LAYOUT_UNKNOWN

Value: 0

Indicates that the layout is unknown, and is likely adapter-dependent.

During creation, the driver chooses the most efficient layout based on other resource properties, especially resource size and flags.

Prefer this choice unless certain functionality is required from another texture layout.

Zero-copy texture upload optimizations exist for UMA architectures; see

[ID3D12Resource::WriteToSubresource](#).

D3D12_TEXTURE_LAYOUT_ROW_MAJOR

Value: 1

Indicates that data for the texture is stored in row-major order (sometimes called "pitch-linear order").

This texture layout locates consecutive texels of a row contiguously in memory, before the texels of the next row.

Similarly, consecutive texels of a particular depth or array slice are contiguous in memory before the texels of the next depth or array slice.

Padding may exist between rows and between depth or array slices to align collections of data.

A stride is the distance in memory between rows, depth, or array slices; and it includes any padding.

This texture layout enables sharing of the texture data between multiple adapters, when other layouts aren't available.

Many restrictions apply, because this layout is generally not efficient for extensive usage:

- The locality of nearby texels is not rotationally invariant.
- Only the following texture properties are supported:
 - [D3D12_RESOURCE_DIMENSION_TEXTURE_2D](#).
 - A single mip level.
 - A single array slice.
 - 64KB alignment.
 - Non-MSAA.
 - No [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#).
 - The format cannot be a YUV format.
- The texture must be created on a heap with [D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER](#).

Buffers are created with [D3D12_TEXTURE_LAYOUT_ROW_MAJOR](#), because row-major texture data can be located in them without creating a texture object.

This is commonly used for uploading or reading back texture data, especially for discrete/NUMA adapters.

However, [D3D12_TEXTURE_LAYOUT_ROW_MAJOR](#) can also be used when marshaling texture data between GPUs or adapters.

For examples of usage with [ID3D12GraphicsCommandList::CopyTextureRegion](#), see some of the following topics:

- [Default Texture Mapping and Standard Swizzle](#)
- [Predication](#)

- [Multi-engine synchronization](#)
- [Uploading Texture Data](#)

`D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE`

Value: 2

Indicates that the layout within 64KB tiles and tail mip packing is up to the driver.

No standard swizzle pattern.

This texture layout is arranged into contiguous 64KB regions, also known as tiles, containing near equilateral amount of consecutive number of texels along each dimension.

Tiles are arranged in row-major order.

While there is no padding between tiles, there are typically unused texels within the last tile in each dimension.

The layout of texels within the tile is undefined.

Each subresource immediately follows where the previous subresource end, and the subresource order follows the same sequence as subresource ordinals.

However, tail mip packing is adapter-specific.

For more details, see tiled resource tier and [ID3D12Device::GetResourceTiling](#).

This texture layout enables partially resident or sparse texture scenarios when used together with virtual memory page mapping functionality.

This texture layout must be used together with [ID3D12Device::CreateReservedResource](#) to enable the usage of [ID3D12CommandQueue::UpdateTileMappings](#).

Some restrictions apply to textures with this layout:

- The adapter must support [D3D12_TILED_RESOURCES_TIER](#) 1 or greater.
- 64KB alignment must be used.
- [D3D12_RESOURCE_DIMENSION_TEXTURE1D](#) is not supported, nor are all formats.
- The tiled resource tier indicates whether textures with [D3D12_RESOURCE_DIMENSION_TEXTURE3D](#) is supported.

`D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE`

Value: 3

Indicates that a default texture uses the standardized swizzle pattern.

This texture layout is arranged the same way that

[D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE](#) is, except that the layout of texels within

the tile is defined. Tail mip packing is adapter-specific.

This texture layout enables optimizations when marshaling data between multiple adapters or between the CPU and GPU.

The amount of copying can be reduced when multiple components understand the texture memory layout.

This layout is generally more efficient for extensive usage than row-major layout, due to the rotationally invariant locality of neighboring texels.

This layout can typically only be used with adapters that support standard swizzle, but exceptions exist for cross-adapter shared heaps.

The restrictions for this layout are that the following aren't supported:

- [D3D12_RESOURCE_DIMENSION_TEXTURE1D](#)
- Multi-sample anti-aliasing (MSAA)
- [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#)
- Formats within the [DXGI_FORMAT_R32G32B32_TYPELESS](#) group

Remarks

This enum is used by the [D3D12_RESOURCE_DESC](#) structure.

This enumeration controls the swizzle pattern of default textures and enable map support on default textures. Callers must query

[D3D12_FEATURE_DATA_D3D12_OPTIONS](#) to ensure that each option is supported.

The standard swizzle formats applies within each page-sized chunk, and pages are laid out in linear order with respect to one another. A 16-bit interleave pattern defines the conversion from pre-swizzled intra-page location to the post-swizzled location.

2D Texture Standard Swizzle Pattern

8bpp	XYXY XYXY YYYY XXXX		
16bpp	XYXY XYXY YYYY XXX-	32bpp	XYXY XYXY YYYY XX--
64bpp	XYXY XYXY XXYY X---	128bpp	XYXY XYXY XXYY ----

3D Texture Standard Swizzle Pattern

8bpp	XYZX YYZZ ZZYY XXXX		
16bpp	XYZX YYZZ ZZYY XXX-	32bpp	XYZX YZXY ZZYY XX--
64bpp	XYZX YZXX ZZYY X---	128bpp	XYZX YZXX ZZYY ----

To demonstrate, consider the 2D 32bpp swizzle format above. This is represented by the following interleave masks, where bits on the left are most-significant:

syntax

```
UINT xBytesMask = 1010 1010 1000 1111  
UINT yMask = 0101 0101 0111 0000
```

To compute the swizzled address, the following code could be used (where the `_pdep_u32` intrinsic instruction is supported):

syntax

```
UINT swizzledOffset = resourceBaseOffset +  
    _pdep_u32(xOffset, xBytesMask) +  
    _pdep_u32(yOffset, yBytesMask);
```

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Enumerations](#)

[UMA Optimizations: CPU Accessible Textures and Standard Swizzle](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILE_COPY_FLAGS enumeration (d3d12.h)

Article01/31/2022

Specifies how to copy a tile.

Syntax

C++

```
typedef enum D3D12_TILE_COPY_FLAGS {
    D3D12_TILE_COPY_FLAG_NONE = 0,
    D3D12_TILE_COPY_FLAG_NO_HAZARD = 0x1,
    D3D12_TILE_COPY_FLAG_LINEAR_BUFFER_TO_SWIZZLED_TILED_RESOURCE = 0x2,
    D3D12_TILE_COPY_FLAG_SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER = 0x4
};
```

Constants

[+] Expand table

D3D12_TILE_COPY_FLAG_NONE

Value: 0

No tile-copy flags are specified.

D3D12_TILE_COPY_FLAG_NO_HAZARD

Value: 0x1

Indicates that the GPU isn't currently referencing any of the portions of destination memory being written.

D3D12_TILE_COPY_FLAG_LINEAR_BUFFER_TO_SWIZZLED_TILED_RESOURCE

Value: 0x2

Indicates that the [ID3D12GraphicsCommandList::CopyTiles](#) operation involves copying a linear buffer to a swizzled tiled resource. This means to copy tile data from the specified buffer location, reading tiles sequentially, to the specified tile region (in x,y,z order if the region is a box), swizzling to optimal hardware memory layout as needed.

In this [ID3D12GraphicsCommandList::CopyTiles](#) call, you specify the source data with the *pBuffer* parameter and the destination with the *pTiledResource* parameter.

D3D12_TILE_COPY_FLAG_SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER

Value: 0x4

Indicates that the [ID3D12GraphicsCommandList::CopyTiles](#) operation involves copying a swizzled tiled resource to a linear buffer. This means to copy tile data from the tile region, reading tiles sequentially (in x,y,z order if the region is a box),

to the specified buffer location, deswizzling to linear memory layout as needed.

In this [ID3D12GraphicsCommandList::CopyTiles](#) call, you specify the source data with the *pTiledResource* parameter and the destination with the *pBuffer* parameter.

Remarks

This enum is used by the [CopyTiles](#) method.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILE_MAPPING_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies how to perform a tile-mapping operation.

Syntax

C++

```
typedef enum D3D12_TILE_MAPPING_FLAGS {
    D3D12_TILE_MAPPING_FLAG_NONE = 0,
    D3D12_TILE_MAPPING_FLAG_NO_HAZARD = 0x1
};
```

Constants

 Expand table

<code>D3D12_TILE_MAPPING_FLAG_NONE</code>

Value: *0*

No tile-mapping flags are specified.

<code>D3D12_TILE_MAPPING_FLAG_NO_HAZARD</code>
--

Value: *0x1*

Unsupported, do not use.

Remarks

This enum is used by the following methods:

- [CopyTileMappings](#)
- [UpdateTileMappings](#)

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILE_RANGE_FLAGS enumeration (d3d12.h)

Article 02/22/2024

Specifies a range of tile mappings.

Syntax

C++

```
typedef enum D3D12_TILE_RANGE_FLAGS {
    D3D12_TILE_RANGE_FLAG_NONE = 0,
    D3D12_TILE_RANGE_FLAG_NULL = 1,
    D3D12_TILE_RANGE_FLAG_SKIP = 2,
    D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE = 4
};
```

Constants

[] Expand table

`D3D12_TILE_RANGE_FLAG_NONE`

Value: 0

No tile-mapping flags are specified.

`D3D12_TILE_RANGE_FLAG_NULL`

Value: 1

The tile range is **NULL**.

`D3D12_TILE_RANGE_FLAG_SKIP`

Value: 2

Skip the tile range.

`D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE`

Value: 4

Reuse a single tile in the tile range.

Remarks

Use these flags with [ID3D12CommandQueue::UpdateTileMappings](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILE_REGION_SIZE structure (d3d12.h)

Article02/22/2024

Describes the size of a tiled region.

Syntax

C++

```
typedef struct D3D12_TILE_REGION_SIZE {
    UINT    NumTiles;
    BOOL    UseBox;
    UINT    Width;
    UINT16  Height;
    UINT16  Depth;
} D3D12_TILE_REGION_SIZE;
```

Members

NumTiles

The number of tiles in the tiled region.

UseBox

Specifies whether the runtime uses the **Width**, **Height**, and **Depth** members to define the region.

If **TRUE**, the runtime uses the **Width**, **Height**, and **Depth** members to define the region. In this case, **NumTiles** should be equal to **Width * Height * Depth**.

If **FALSE**, the runtime ignores the **Width**, **Height**, and **Depth** members and uses the **NumTiles** member to traverse tiles in the resource linearly across x, then y, then z (as applicable) and then spills over mipmaps/arrays in subresource order. For example, use this technique to map an entire resource at once.

Regardless of whether you specify **TRUE** or **FALSE** for **UseBox**, you use a [D3D12_TILED_RESOURCE_COORDINATE](#) structure to specify the starting location for the region within the resource as a separate parameter outside of this structure by using x, y, and z coordinates.

When the region includes mipmaps that are packed with nonstandard tiling, **UseBox** must be **FALSE** because tile dimensions are not standard and the app only knows a count of how many tiles are consumed by the packed area, which is per array slice. The corresponding (separate) starting location parameter uses **x** to offset into the flat range of tiles in this case, and **y** and **z** coordinates must each be 0.

Width

The width of the tiled region, in tiles. Used for buffer and 1D, 2D, and 3D textures.

Height

The height of the tiled region, in tiles. Used for 2D and 3D textures.

Depth

The depth of the tiled region, in tiles. Used for 3D textures or arrays. For arrays, used for advancing in depth jumps to next slice of same mipmap size, which isn't contiguous in the subresource counting space if there are multiple mipmaps.

Remarks

This structure is used by the [CopyTiles](#), [CopyTileMappings](#) and [UpdateTileMappings](#) methods.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_TILE_REGION_SIZE](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_TILE_SHAPE structure (d3d12.h)

Article 02/22/2024

Describes the shape of a tile by specifying its dimensions.

Syntax

C++

```
typedef struct D3D12_TILE_SHAPE {
    UINT WidthInTexels;
    UINT HeightInTexels;
    UINT DepthInTexels;
} D3D12_TILE_SHAPE;
```

Members

`WidthInTexels`

The width in texels of the tile.

`HeightInTexels`

The height in texels of the tile.

`DepthInTexels`

The depth in texels of the tile.

Remarks

This structure is used by the [GetResourceTiling](#) method.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_TILE_SHAPE](#)

[Core Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILED_RESOURCE_COORDINATE structure (d3d12.h)

Article02/22/2024

Describes the coordinates of a tiled resource.

Syntax

C++

```
typedef struct D3D12_TILED_RESOURCE_COORDINATE {
    UINT X;
    UINT Y;
    UINT Z;
    UINT Subresource;
} D3D12_TILED_RESOURCE_COORDINATE;
```

Members

X

The x-coordinate of the tiled resource.

Y

The y-coordinate of the tiled resource.

Z

The z-coordinate of the tiled resource.

Subresource

The index of the subresource for the tiled resource.

For mipmaps that use nonstandard tiling, or are packed, or both use nonstandard tiling and are packed, any subresource value that indicates any of the packed mipmaps all refer to the same tile. Additionally, the X coordinate is used to indicate a tile within the packed mip region, rather than a logical region of a single subresource. The Y and Z coordinates must be zero.

Remarks

This structure is used by the [CopyTiles](#), [CopyTileMappings](#) and [UpdateTileMappings](#) methods.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[CD3DX12_TILED_RESOURCE_COORDINATE](#)

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_TILED_RESOURCES_TIER enumeration (d3d12.h)

Article01/31/2022

Identifies the tier level at which tiled resources are supported.

Syntax

C++

```
typedef enum D3D12_TILED_RESOURCES_TIER {
    D3D12_TILED_RESOURCES_TIER_NOT_SUPPORTED = 0,
    D3D12_TILED_RESOURCES_TIER_1 = 1,
    D3D12_TILED_RESOURCES_TIER_2 = 2,
    D3D12_TILED_RESOURCES_TIER_3 = 3,
    D3D12_TILED_RESOURCES_TIER_4 = 4
};
```

Constants

[+] Expand table

D3D12_TILED_RESOURCES_TIER_NOT_SUPPORTED

Value: 0

Indicates that textures cannot be created with the [D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE](#) layout.

[ID3D12Device::CreateReservedResource](#) cannot be used, not even for buffers.

D3D12_TILED_RESOURCES_TIER_1

Value: 1

Indicates that 2D textures can be created with the [D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE](#) layout. Limitations exist for certain resource formats and properties. For more details, see [D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE](#).

[ID3D12Device::CreateReservedResource](#) can be used.

GPU reads or writes to NULL mappings are undefined.

Applications are encouraged to workaround this limitation by repeatedly mapping the same page to everywhere a NULL mapping would've been used.

When the size of a texture mipmap level is an integer multiple of the standard tile shape for its format, it is guaranteed to be nonpacked.

D3D12_TILED_RESOURCES_TIER_2

Value: 2

Indicates that a superset of Tier_1 functionality is supported, including this additional support:

- When the size of a texture mipmap level is at least one standard tile shape for its format, the mipmap level is guaranteed to be nonpacked.
For more info, see [D3D12_PACKED_MIP_INFO](#).
- Shader instructions are available for clamping level-of-detail (LOD) and for obtaining status about the shader operation.
For info about one of these shader instructions, see [Sample\(S,float,int,float,uint\)](#).
[Sample\(S,float,int,float,uint\)](#).
- Reading from **NULL**-mapped tiles treat that sampled value as zero.
Writes to **NULL**-mapped tiles are discarded.

Adapters that support feature level 12_0 all support TIER_2 or greater.

D3D12_TILED_RESOURCES_TIER_3

Value: 3

Indicates that a superset of Tier 2 is supported, with the addition that 3D textures ([Volume Tiled Resources](#)) are supported.

D3D12_TILED_RESOURCES_TIER_4

Value: 4

Remarks

This enum is used by the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

There are three discrete pieces of functionality bundled together for tiled resource functionality:

- A tile-based texture layout option where nearby texel addresses contain nearby data coordinates. A tile of texels contains nearly the same amount of texels in each cardinal dimension of the resource. This layout is represented in D3D12 by [D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE](#).
- Reserve a region of virtual address space for a resource, where each page is initially NULL-mapped. In D3D12, this operation is encapsulated within [ID3D12Device::CreateReservedResource](#), which only works with textures that have the D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE layout.
- The ability to change page mappings and manipulate texture data on tile granularities. In D3D12, these operations are [ID3D12CommandQueue::UpdateTileMappings](#), [ID3D12CommandQueue::CopyTileMappings](#), and [ID3D12GraphicsCommandList::CopyTiles](#).

Three significant changes over D3D11 are:

- Tile pools are replaced by heaps. Heaps provide a superset of capabilities than D3D11 tile pools do.
- Reserved resources may be mapped to pages from multiple heaps at the same time. The D3D11 restriction that all non-NULl mapped pages must come from the same heap does not exist.
- Applications should be aware of GPU virtual address capabilities, which enable litmus tests for particular usage scenarios. See [D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

D3D12_TRI_STATE enumeration (d3d12.h)

Article 02/22/2024

Defines constants that specify TBD

Syntax

C++

```
typedef enum D3D12_TRI_STATE {
    D3D12_TRI_STATE_UNKNOWN,
    D3D12_TRI_STATE_FALSE,
    D3D12_TRI_STATE_TRUE
} ;
```

Constants

[] Expand table

	D3D12_TRI_STATE_UNKNOWN Specifies TBD.
	D3D12_TRI_STATE_FALSE Specifies TBD.
	D3D12_TRI_STATE_TRUE Specifies TBD.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_UAV_DIMENSION enumeration (d3d12.h)

Article 02/14/2023

Identifies unordered-access view options.

Syntax

C++

```
typedef enum D3D12_UAV_DIMENSION {
    D3D12_UAV_DIMENSION_UNKNOWN = 0,
    D3D12_UAV_DIMENSION_BUFFER = 1,
    D3D12_UAV_DIMENSION_TEXTURE1D = 2,
    D3D12_UAV_DIMENSION_TEXTURE1DARRAY = 3,
    D3D12_UAV_DIMENSION_TEXTURE2D = 4,
    D3D12_UAV_DIMENSION_TEXTURE2DARRAY = 5,
    D3D12_UAV_DIMENSION_TEXTURE2DMS,
    D3D12_UAV_DIMENSION_TEXTURE2DMSARRAY,
    D3D12_UAV_DIMENSION_TEXTURE3D = 8
};
```

Constants

[] Expand table

	D3D12_UAV_DIMENSION_UNKNOWN Value: 0 The view type is unknown.
	D3D12_UAV_DIMENSION_BUFFER Value: 1 View the resource as a buffer.
	D3D12_UAV_DIMENSION_TEXTURE1D Value: 2 View the resource as a 1D texture.
	D3D12_UAV_DIMENSION_TEXTURE1DARRAY Value: 3 View the resource as a 1D texture array.

`D3D12_UAV_DIMENSION_TEXTURE2D`

Value: 4

View the resource as a 2D texture.

`D3D12_UAV_DIMENSION_TEXTURE2DARRAY`

Value: 5

View the resource as a 2D texture array.

`D3D12_UAV_DIMENSION_TEXTURE3D`

Value: 8

View the resource as a 3D texture array.

Remarks

Specify one of the values in this enumeration in the `ViewDimension` member of a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_UNORDERED_ACCESS_VIEW_DESC

C structure (d3d12.h)

Article 02/22/2024

Describes the subresources from a resource that are accessible by using an unordered-access view.

Syntax

C++

```
typedef struct D3D12_UNORDERED_ACCESS_VIEW_DESC {
    DXGI_FORMAT           Format;
    D3D12_UAV_DIMENSION ViewDimension;
    union {
        D3D12_BUFFER_UAV      Buffer;
        D3D12_TEX1D_UAV       Texture1D;
        D3D12_TEX1D_ARRAY_UAV Texture1DArray;
        D3D12_TEX2D_UAV       Texture2D;
        D3D12_TEX2D_ARRAY_UAV Texture2DArray;
        D3D12_TEX2DMS_UAV     Texture2DMS;
        D3D12_TEX2DMS_ARRAY_UAV Texture2DMSArray;
        D3D12_TEX3D_UAV       Texture3D;
    };
} D3D12_UNORDERED_ACCESS_VIEW_DESC;
```

Members

Format

A [DXGI_FORMAT](#)-typed value that specifies the viewing format.

ViewDimension

A [D3D12_UAV_DIMENSION](#)-typed value that specifies the resource type of the view. This type specifies how the resource will be accessed. This member also determines which _UAV to use in the union below.

Buffer

A [D3D12_BUFFER_UAV](#) structure that specifies which buffer elements can be accessed.

Texture1D

A [D3D12_TEX1D_UAV](#) structure that specifies the subresources in a 1D texture that can be accessed.

Texture1DArray

A [D3D12_TEX1D_ARRAY_UAV](#) structure that specifies the subresources in a 1D texture array that can be accessed.

Texture2D

A [D3D12_TEX2D_UAV](#) structure that specifies the subresources in a 2D texture that can be accessed.

Texture2DArray

A [D3D12_TEX2D_ARRAY_UAV](#) structure that specifies the subresources in a 2D texture array that can be accessed.

Texture2DMS

Texture2DMSArray

Texture3D

A [D3D12_TEX3D_UAV](#) structure that specifies subresources in a 3D texture that can be accessed.

Remarks

Pass an unordered-access-view description into [ID3D12Device::CreateUnorderedAccessView](#) to create a view.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VARIABLE_SHADING_RATE_TIER enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a shading rate tier (for variable-rate shading, or VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

C++

```
typedef enum D3D12_VARIABLE_SHADING_RATE_TIER {
    D3D12_VARIABLE_SHADING_RATE_TIER_NOT_SUPPORTED = 0,
    D3D12_VARIABLE_SHADING_RATE_TIER_1 = 1,
    D3D12_VARIABLE_SHADING_RATE_TIER_2 = 2
};
```

Constants

[] [Expand table](#)

<code>D3D12_VARIABLE_SHADING_RATE_TIER_NOT_SUPPORTED</code>

Value: 0

Specifies that variable-rate shading is not supported.

<code>D3D12_VARIABLE_SHADING_RATE_TIER_1</code>

Value: 1

Specifies that variable-rate shading tier 1 is supported.

<code>D3D12_VARIABLE_SHADING_RATE_TIER_2</code>

Value: 2

Specifies that variable-rate shading tier 2 is supported.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure (d3d12.h)

Article 02/22/2024

Represents versioned Device Removed Extended Data (DRED) data, so that debuggers and debugger extensions can access DRED data.

Syntax

C++

```
typedef struct D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA {
    D3D12_DRED_VERSION Version;
    union {
        D3D12_DEVICE_REMOVED_EXTENDED_DATA Dred_1_0;
        D3D12_DEVICE_REMOVED_EXTENDED_DATA1 Dred_1_1;
        D3D12_DEVICE_REMOVED_EXTENDED_DATA2 Dred_1_2;
        D3D12_DEVICE_REMOVED_EXTENDED_DATA3 Dred_1_3;
    };
} D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA;
```

Members

Version

A [D3D12_DRED_VERSION](#) value, specifying a DRED version. This value determines which inner data member (of the union) is active.

Dred_1_0

A [D3D12_DEVICE_REMOVED_EXTENDED_DATA](#) value, containing DRED version 1.0 data.

Dred_1_1

A [D3D12_DEVICE_REMOVED_EXTENDED_DATA1](#) value, containing DRED version 1.1 data.

Dred_1_2

Dred_1_3

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

- [Core structures](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_VERSIONED_ROOT_SIGNATURE_DESC structure (d3d12.h)

Article 02/22/2024

Holds any version of a root signature description, and is designed to be used with serialization/deserialization functions.

Syntax

C++

```
typedef struct D3D12_VERSIONED_ROOT_SIGNATURE_DESC {
    D3D_ROOT_SIGNATURE_VERSION Version;
    union {
        D3D12_ROOT_SIGNATURE_DESC Desc_1_0;
        D3D12_ROOT_SIGNATURE_DESC1 Desc_1_1;
        D3D12_ROOT_SIGNATURE_DESC2 Desc_1_2;
    };
} D3D12_VERSIONED_ROOT_SIGNATURE_DESC;
```

Members

Version

Specifies one member of D3D_ROOT_SIGNATURE_VERSION that determines the contents of the union.

Desc_1_0

Specifies a [D3D12_ROOT_SIGNATURE_DESC](#) (version 1.0).

Desc_1_1

Specifies a [D3D12_ROOT_SIGNATURE_DESC1](#) (version 1.1).

Desc_1_2

Remarks

Use this structure with the following methods.

- [GetRootSignatureDescAtVersion](#)

- [GetUnconvertedRootSignatureDesc](#)
- [D3D12SerializeVersionedRootSignature](#)

Refer to the helper structure [CD3DX12_VERSIONED_ROOT_SIGNATURE_DESC](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VERTEX_BUFFER_VIEW structure (d3d12.h)

Article02/22/2024

Describes a vertex buffer view.

Syntax

C++

```
typedef struct D3D12_VERTEX_BUFFER_VIEW {
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation;
    UINT                     SizeInBytes;
    UINT                     StrideInBytes;
} D3D12_VERTEX_BUFFER_VIEW;
```

Members

BufferLocation

Specifies a D3D12_GPU_VIRTUAL_ADDRESS that identifies the address of the buffer.

SizeInBytes

Specifies the size in bytes of the buffer.

StrideInBytes

Specifies the size in bytes of each vertex entry.

Remarks

Use this structure with the [IASetVertexBuffers](#) method.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VIEW_INSTANCE_LOCATION structure (d3d12.h)

Article04/02/2021

Specifies the viewport/stencil and render target associated with a view instance.

Syntax

C++

```
typedef struct D3D12_VIEW_INSTANCE_LOCATION {
    UINT ViewportArrayIndex;
    UINT RenderTargetArrayIndex;
} D3D12_VIEW_INSTANCE_LOCATION;
```

Members

ViewportArrayIndex

The index of the viewport in the viewports array to be used by the view instance associated with this location.

RenderTargetArrayIndex

The index of the render target in the render targets array to be used by the view instance associated with this location.

Remarks

The values specified in a view instance location structure can be added to ViewportArrayIndex and RenderTargetArrayIndex values output by the shader prior to rasterization to compute the final effective index of the viewport and render target to send primitives to. If a computed index is out of range (that is, when the index is larger than the number of viewport or render target elements in their respective arrays) then the effective index becomes 0.

For shaders that dynamically select the viewport or render target indices, an application can set all the view instance locations declared in a PSO to the same value to act as a uniform base value for all views.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VIEW_INSTANCING_DESC structure (d3d12.h)

Article04/02/2021

Specifies parameters used during view instancing configuration.

Syntax

C++

```
typedef struct D3D12_VIEW_INSTANCING_DESC {
    UINT ViewInstanceCount;
    const D3D12_VIEW_INSTANCE_LOCATION *pViewInstanceLocations;
    D3D12_VIEW_INSTANCING_FLAGS Flags;
} D3D12_VIEW_INSTANCING_DESC;
```

Members

ViewInstanceCount

Specifies the number of views to be used, up to D3D12_MAX_VIEW_INSTANCE_COUNT.

pViewInstanceLocations

The address of a memory location that contains **ViewInstanceCount** view instance location structures that specify the location of viewport/scissor and render target details of each view instance.

Flags

Configures view instancing with additional options.

Remarks

View instancing is declared in a PSO using this structure. The view instance count is set in the PSO to allow whole-pipeline optimization based on the number of views.

View instancing is disabled when it's not declared in the PSO or when ViewInstanceCount is set to 0. When disabled, rendering behaves as if view instancing is enabled and ViewInstanceCount is set to 1; shaders only see a value of 0 in SV_ViewID

and just one view instance is produced. This allows shaders that are aware of view instancing to still be used in PSOs that disable it. Some adapters might support shader model 6.1 (which exposes SV_ViewID) but not view instancing; these adapters must still support shaders that input SV_ViewID in PSOs that declare ViewInstanceCount as 0 or 1.

The shader prior to rasterization can output SV_RenderTargetArrayIndex and SV_ViewportArrayIndex values which don't have to depend on SV_ViewID. To compute the final effective index of the viewport and render target where primitives will be sent, these values, when present, are added to the ViewportArrayIndex and RenderTargetArrayIndex values of the view instance locations declared in the PSO. If a computed index is out of range (that is, when the index is larger than the number of viewport or render target elements in their respective arrays) then the effective index becomes 0.

For shaders that dynamically select the viewport or render target indices, an application can set all the view instance locations declared in the PSO to a single value (such as 0) to act as a uniform base index to which the dynamically-selected SV_RenderTargetArrayIndex and SV_ViewportArrayIndex values are added.

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VIEW_INSTANCING_FLAGS enumeration (d3d12.h)

Article02/22/2024

Specifies options for view instancing.

Syntax

C++

```
typedef enum D3D12_VIEW_INSTANCING_FLAGS {
    D3D12_VIEW_INSTANCING_FLAG_NONE = 0,
    D3D12_VIEW_INSTANCING_FLAG_ENABLE_VIEW_INSTANCE_MASKING = 0x1
} ;
```

Constants

[+] Expand table

<code>D3D12_VIEW_INSTANCING_FLAG_NONE</code>	Value: <i>0</i> Indicates a default view instancing configuration.
<code>D3D12_VIEW_INSTANCING_FLAG_ENABLE_VIEW_INSTANCE_MASKING</code>	Value: <i>0x1</i> Enables view instance masking.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_VIEW_INSTANCING_TIER enumeration (d3d12.h)

Article 01/31/2022

Indicates the tier level at which view instancing is supported.

Syntax

C++

```
typedef enum D3D12_VIEW_INSTANCING_TIER {
    D3D12_VIEW_INSTANCING_TIER_NOT_SUPPORTED = 0,
    D3D12_VIEW_INSTANCING_TIER_1 = 1,
    D3D12_VIEW_INSTANCING_TIER_2 = 2,
    D3D12_VIEW_INSTANCING_TIER_3 = 3
};
```

Constants

[] Expand table

`D3D12_VIEW_INSTANCING_TIER_NOT_SUPPORTED`

Value: 0

View instancing is not supported.

`D3D12_VIEW_INSTANCING_TIER_1`

Value: 1

View instancing is supported by draw-call level looping only.

`D3D12_VIEW_INSTANCING_TIER_2`

Value: 2

View instancing is supported by draw-call level looping at worst, but the GPU can perform view instancing more efficiently in certain circumstances which are architecture-dependent.

`D3D12_VIEW_INSTANCING_TIER_3`

Value: 3

View instancing is supported and instancing begins with the first shader stage that references `SV_ViewID` or with rasterization if no shader stage references `SV_ViewID`. This means that redundant work is eliminated across view instances when it's not dependent on `SV_ViewID`. Before rasterization, work that doesn't directly depend on `SV_ViewID` is shared across all views; only work

that depends on SV_ViewID is repeated for each view.

Note If a hull shader produces tessellation factors that are dependent on SV_ViewID, then tessellation and all subsequent work must be repeated per-view. Similarly, if the amount of geometry produced by the geometry shader depends on SV_ViewID, then the geometry shader must be repeated per-view before proceeding to rasterization.

View instance masking only effects whether work that directly depends on SV_ViewID is performed, not the entire loop iteration (per-view). If the view instance mask is non-0, some work that depends on SV_ViewID might still be performed on masked-off pixels but will have no externally-visible effect; for example, no UAV writes are performed and clipping/rasterization is not invoked. If the view instance mask is 0 no work is performed, including work that's not dependent on SV_ViewID.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_VIEWPORT structure (d3d12.h)

Article 02/22/2024

Describes the dimensions of a viewport.

Syntax

C++

```
typedef struct D3D12_VIEWPORT {
    FLOAT TopLeftX;
    FLOAT TopLeftY;
    FLOAT Width;
    FLOAT Height;
    FLOAT MinDepth;
    FLOAT MaxDepth;
} D3D12_VIEWPORT;
```

Members

TopLeftX

X position of the left hand side of the viewport.

TopLeftY

Y position of the top of the viewport.

Width

Width of the viewport.

Height

Height of the viewport.

MinDepth

Minimum depth of the viewport. Ranges between 0 and 1.

MaxDepth

Maximum depth of the viewport. Ranges between 0 and 1.

Remarks

Pass an array of these structures to the *pViewports* parameter in a call to [ID3D12GraphicsCommandList::RSSetViewports](#) to set viewports for the display.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_WAVE_MMA_TIER enumeration (d3d12.h)

Article02/22/2024

Defines constants that specify a level of support for WaveMMA (wave_matrix) operations.

Syntax

C++

```
typedef enum D3D12_WAVE_MMA_TIER {
    D3D12_WAVE_MMA_TIER_NOT_SUPPORTED = 0,
    D3D12_WAVE_MMA_TIER_1_0 = 10
};
```

Constants

[] Expand table

D3D12_WAVE_MMA_TIER_NOT_SUPPORTED	Value: 0 Specifies that WaveMMA (wave_matrix) operations are not supported.
D3D12_WAVE_MMA_TIER_1_0	Value: 10 Specifies that WaveMMA (wave_matrix) operations are supported.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_WRITEBUFFERIMMEDIATE_MODE enumeration (d3d12.h)

Article 01/31/2022

Specifies the mode used by a **WriteBufferImmediate** operation.

Syntax

C++

```
typedef enum D3D12_WRITEBUFFERIMMEDIATE_MODE {
    D3D12_WRITEBUFFERIMMEDIATE_MODE_DEFAULT = 0,
    D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_IN = 0x1,
    D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_OUT = 0x2
};
```

Constants

[Expand table](#)

`D3D12_WRITEBUFFERIMMEDIATE_MODE_DEFAULT`

Value: *0*

The write operation behaves the same as normal copy-write operations.

`D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_IN`

Value: *0x1*

The write operation is guaranteed to occur after all preceding commands in the command stream have started, including previous **WriteBufferImmediate** operations.

`D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_OUT`

Value: *0x2*

The write operation is deferred until all previous commands in the command stream have completed through the GPU pipeline, including previous **WriteBufferImmediate** operations. Write operations that specify `D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_OUT` don't block subsequent operations from starting. If there are no previous operations in the command stream, then the write operation behaves as if `D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_IN` was specified.

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

See also

[Core Enumerations](#)

[ID3D12GraphicsCommandList::WriteBufferImmediate](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_WRITEBUFFERIMMEDIATE_PARA METER structure (d3d12.h)

Article 02/22/2024

Specifies the immediate value and destination address written using [ID3D12GraphicsCommandList2::WriteBufferImmediate](#).

Syntax

C++

```
typedef struct D3D12_WRITEBUFFERIMMEDIATE_PARAMETER {
    D3D12_GPU_VIRTUAL_ADDRESS Dest;
    UINT32                  Value;
} D3D12_WRITEBUFFERIMMEDIATE_PARAMETER;
```

Members

Dest

The GPU virtual address at which to write the value. The address must be aligned to a 32-bit (4-byte) boundary.

Value

The 32-bit value to write.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12.h

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12CreateDevice function (d3d12.h)

Article 07/27/2022

Creates a device that represents the display adapter.

Syntax

C++

```
HRESULT D3D12CreateDevice(
    [in, optional] IUnknown* pAdapter,
    D3D_FEATURE_LEVEL MinimumFeatureLevel,
    [in] REFIID riid,
    [out, optional] void** ppDevice
);
```

Parameters

[in, optional] pAdapter

Type: **IUnknown***

A pointer to the video adapter to use when creating a [device](#). Pass **NULL** to use the default adapter, which is the first adapter that is enumerated by [IDXGIFactory1::EnumAdapters](#).

Note Don't 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.

MinimumFeatureLevel

Type: **D3D_FEATURE_LEVEL**

The minimum **D3D_FEATURE_LEVEL** required for successful device creation.

[in] riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the device interface. This parameter, and *ppDevice*, can be addressed with the single macro [IID_PPV_ARGS](#).

[out, optional] ppDevice

Type: **void****

A pointer to a memory block that receives a pointer to the device. Pass **NULL** to test if device creation would succeed, but to not actually create the device. If **NULL** is passed and device creation would succeed, **S_FALSE** is returned.

Return value

Type: **HRESULT**

This method can return one of the [Direct3D 12 Return Codes](#).

Possible return values include those documented for [CreateDXGIFactory1](#) and [IDXGIFactory::EnumAdapters](#).

If **ppDevice** is **NULL** and the function succeeds, **S_FALSE** is returned, rather than **S_OK**.

Remarks

Direct3D 12 devices are singletons per adapter. If a Direct3D 12 device already exists in the current process for a given adapter, then a subsequent call to [D3D12CreateDevice](#) returns the existing device. If the current Direct3D 12 device is in a removed state (that is, [ID3D12Device::GetDeviceRemovedReason](#) returns a failing HRESULT), then [D3D12CreateDevice](#) fails instead of returning the existing device. The sameness of two adapters (that is, they have the same identity) is determined by comparing their LUIDs, not their pointers.

In order to be sure to pick up the first adapter that supports D3D12, use the following code.

syntax

```
void GetHardwareAdapter(IDXGIFactory4* pFactory, IDXGIAdapter1** ppAdapter)
{
    *ppAdapter = nullptr;
    for (UINT adapterIndex = 0; ; ++adapterIndex)
    {
        IDXGIAdapter1* pAdapter = nullptr;
        if (DXGI_ERROR_NOT_FOUND == pFactory->EnumAdapters1(adapterIndex,
&pAdapter))
        {
            // No more adapters to enumerate.
            break;
        }
    }
}
```

```

        // Check to see if the adapter supports Direct3D 12, but don't
create the
        // actual device yet.
        if (SUCCEEDED(D3D12CreateDevice(pAdapter, D3D_FEATURE_LEVEL_11_0,
__uuidof(ID3D12Device), nullptr)))
        {
            *ppAdapter = pAdapter;
            return;
        }
        pAdapter->Release();
    }
}

```

The function signature PFN_D3D12_CREATE_DEVICE is provided as a typedef, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

The **REFIID**, or **GUID**, of the interface to a device can be obtained by using the `__uuidof()` macro. For example, `__uuidof(ID3D12Device)` will get the **GUID** of the interface to a device.

Examples

Create a hardware based device, unless instructed to create a WARP software device.

C++

```

ComPtr<IDXGIFactory4> factory;
ThrowIfFailed(CreateDXGIFactory1(IID_PPV_ARGS(&factory)));

if (m_useWarpDevice)
{
    ComPtr<IDXGIAdapter> warpAdapter;
    ThrowIfFailed(factory->EnumWarpAdapter(IID_PPV_ARGS(&warpAdapter)));

    ThrowIfFailed(D3D12CreateDevice(
        warpAdapter.Get(),
        D3D_FEATURE_LEVEL_11_0,
        IID_PPV_ARGS(&m_device)
    ));
}
else
{
    ComPtr<IDXGIAdapter1> hardwareAdapter;
    GetHardwareAdapter(factory.Get(), &hardwareAdapter);

    ThrowIfFailed(D3D12CreateDevice(
        hardwareAdapter.Get(),
        D3D_FEATURE_LEVEL_11_0,

```

```
IID_PPV_ARGS(&m_device)
));
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Working Samples](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12CreateRootSignatureDeserializer function (d3d12.h)

Article 02/22/2024

Deserializes a root signature so you can determine the layout definition ([D3D12_ROOT_SIGNATURE_DESC](#)).

Syntax

C++

```
HRESULT D3D12CreateRootSignatureDeserializer(
    [in]  LPCVOID pSrcData,
    [in]  SIZE_T   SrcDataSizeInBytes,
    [in]  REFIID   pRootSignatureDeserializerInterface,
    [out] void     **ppRootSignatureDeserializer
);
```

Parameters

[in] `pSrcData`

Type: `LPCVOID`

A pointer to the source data for the serialized root signature.

[in] `SrcDataSizeInBytes`

Type: `SIZE_T`

The size, in bytes, of the block of memory that `pSrcData` points to.

[in] `pRootSignatureDeserializerInterface`

Type: `REFIID`

The globally unique identifier ([GUID](#)) for the root signature deserializer interface. See remarks.

[out] `ppRootSignatureDeserializer`

Type: `void**`

A pointer to a memory block that receives a pointer to the root signature deserializer.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

Remarks

This function has been superceded by [D3D12CreateVersionedRootSignatureDeserializer](#).

If an application has a serialized root signature already or has a compiled shader that contains a root signature and wants to determine the layout definition, it can call [D3D12CreateRootSignatureDeserializer](#) to generate a [ID3D12RootSignatureDeserializer](#) interface. [ID3D12RootSignatureDeserializer::GetRootSignature](#) can return the serialized data structure ([D3D12_ROOT_SIGNATURE_DESC](#)).

[ID3D12RootSignatureDeserializer](#) just owns the lifetime of the memory for the serialized data structure.

The [REFIID](#), or [GUID](#), of the interface to the root signature deserializer can be obtained by using the [_uuidof\(\)](#) macro. For example, [_uuidof\(ID3D12RootSignatureDeserializer\)](#) will get the [GUID](#) of the interface to a root signature deserializer.

The function signature [PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER](#) is provided as a [typedef](#), so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Creating a Root Signature](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12CreateVersionedRootSignatureDeserializer function (d3d12.h)

Article02/22/2024

Generates an interface that can return the deserialized data structure, via [GetUnconvertedRootSignatureDesc](#).

Syntax

C++

```
HRESULT D3D12CreateVersionedRootSignatureDeserializer(
    [in]    LPCVOID pSrcData,
    [in]    SIZE_T   SrcDataSizeInBytes,
    [in]    REFIID   pRootSignatureDeserializerInterface,
    [out]   void     **ppRootSignatureDeserializer
);
```

Parameters

[in] `pSrcData`

Type: `LPCVOID`

A pointer to the source data for the serialized root signature.

[in] `SrcDataSizeInBytes`

Type: `SIZE_T`

The size, in bytes, of the block of memory that *pSrcData* points to.

[in] `pRootSignatureDeserializerInterface`

Type: `REFIID`

The globally unique identifier (**GUID**) for the root signature deserializer interface. See remarks.

[out] `ppRootSignatureDeserializer`

Type: `void**`

A pointer to a memory block that receives a pointer to the root signature deserializer.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

Remarks

If an application has a serialized root signature already or has a compiled shader that contains a root signature and wants to determine the layout definition, it can call

[D3D12CreateVersionedRootSignatureDeserializer](#) to generate a

[ID3D12VersionedRootSignatureDeserializer](#) interface.

[ID3D12VersionedRootSignatureDeserializer::GetRootSignatureDescAtVersion](#) can return the serialized data structure ([D3D12_ROOT_SIGNATURE_DESC1](#)).

[ID3D12VersionedRootSignatureDeserializer](#) just owns the lifetime of the memory for the serialized data structure.

The **REFIID**, or **GUID**, of the interface to the root signature deserializer can be obtained by using the `_uuidof()` macro. For example,

`_uuidof(ID3D12VersionedRootSignatureDeserializer)` will get the **GUID** of the interface to a root signature deserializer.

The function signature `PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER` is provided as a `typedef`, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

This function supercedes [D3D12CreateRootSignatureDeserializer](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[Core Functions](#)

[Creating a Root Signature](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12EnableExperimentalFeatures function (d3d12.h)

Article 08/19/2022

Enables a list of experimental features.

Syntax

C++

```
HRESULT D3D12EnableExperimentalFeatures(
    UINT      NumFeatures,
    [in] const IID *pIIDs,
    [in] void    *pConfigurationStructs,
    [in] UINT    *pConfigurationStructSizes
);
```

Parameters

NumFeatures

Type: **UINT**

The number of experimental features to enable.

[in] pIIDs

Type: **const IID***

SAL: `__in_ecount(NumFeatures)`

A pointer to an array of IDs that specify which of the available experimental features to enable.

[in] pConfigurationStructs

Type: **void***

SAL: `__in_ecount(NumFeatures)`

Structures that contain additional configuration details that some experimental features might need to be enabled.

[in] pConfigurationStructSizes

Type: **UINT***

SAL: `__in_ecount(NumFeatures)`

The sizes of any configuration structs passed in pConfigurationStructs parameter.

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code that can include E_NOINTERFACE if an unrecognized feature is specified or Developer Mode is not enabled, or E_INVALIDARG if the configuration of a feature is incorrect, the experimental features specified are not compatible, or other errors.

Remarks

Call this function before device creation.

Because the set of experimental features will change over time, and because these features may not be stable, they are intended for development and experimentation only. This is enforced by requiring Developer Mode to be active before any experimental features can be enabled.

The set of experimental features that are currently supported can be found in the D3D12.h header, near the definition of the D3D12EnableExperimentalFeatures function; because experimental features are only made available infrequently, it's typical to find that no experimental features are currently supported.

Some experimental features might be identified by using an IID as the GUID. For these features, you can use D3D12GetDebugInterface, passing an IID as a parameter, to retrieve the interface for manipulating that feature.

If this function is called again with a different list of features to enable, all current D3D12 devices are set to the DEVICE_REMOVED state.

Examples

This example shows what an experimental feature definition looks like.

C++

```

// -----
// Experimental Feature: D3D12ExperimentalShaderModels
// 
// Use with D3D12EnableExperimentalFeatures to enable experimental shader
model support,
// meaning shader models that haven't been finalized for use in retail.
// 
// Enabling D3D12ExperimentalShaderModels needs no configuration struct,
// pass NULL in the pConfigurationStructs array.
// 
// -----
static const UUID D3D12ExperimentalShaderModels = { /* 76f5573e-f13a-40f5-
b297-81ce9e18933f */
    0x76f5573e,
    0xf13a,
    0x40f5,
    { 0xb2, 0x97, 0x81, 0xce, 0x9e, 0x18, 0x93, 0x3f }
};

```

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

Feedback

Was this page helpful?

[] Yes

[] No

D3D12GetDebugInterface function (d3d12.h)

Article 10/13/2021

Gets a debug interface.

Use [D3D12GetInterface](#) to directly access newer interfaces, especially downlevel.

Syntax

C++

```
HRESULT D3D12GetDebugInterface(
    [in]             REFIID riid,
    [out, optional] void    **ppvDebug
);
```

Parameters

[in] riid

Type: [REFIID](#)

The globally unique identifier ([GUID](#)) for the debug interface. The [REFIID](#), or [GUID](#), of the debug interface can be obtained by using the [_uuidof\(\)](#) macro. For example, [_uuidof\(ID3D12Debug\)](#) will get the [GUID](#) of the debug interface.

[out, optional] ppvDebug

Type: [void**](#)

The debug interface, as a pointer to pointer to void. See [ID3D12Debug](#) and [ID3D12DebugDevice](#).

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

The function signature PFN_D3D12_GET_DEBUG_INTERFACE is provided as a typedef, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

Examples

Enable the D3D12 debug layer.

C++

```
// Enable the D3D12 debug layer.  
{  
  
    ComPtr<ID3D12Debug> debugController;  
    if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController))))  
    {  
        debugController->EnableDebugLayer();  
    }  
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12GetInterface function (d3d12.h)

Article 02/22/2024

Selects an SDK version at runtime when the system is in Windows Developer Mode. Supports debug, tools, [DRED](#), and SDK configuration interfaces.

Syntax

C++

```
HRESULT D3D12GetInterface(
    REFCLSID rclsid,
    REFIID    riid,
    void      **ppvDebug
);
```

Parameters

rclsid

Type: `_In_ REFCLSID`

The CLSID associated with the data and code that will be used to create the object.

The following CLSIDs are defined.

- `DEFINE_GUID(CLSID_D3D12Debug, 0xf2352aeb, 0xdd84, 0x49fe, 0xb9, 0x7b, 0xa9, 0xdc, 0xfd, 0xcc, 0x1b, 0x4f);`
- `DEFINE_GUID(CLSID_D3D12Tools, 0xe38216b1, 0x3c8c, 0x4833, 0xaa, 0x09, 0xa, 0x06, 0xb6, 0x5d, 0x96, 0xc8);`
- `DEFINE_GUID(CLSID_D3D12DeviceRemovedExtendedData, 0x4a75bbc4, 0x9ff4, 0x4ad8, 0x9f, 0x18, 0xab, 0xae, 0x84, 0xdc, 0x5f, 0xf2);`
- `DEFINE_GUID(CLSID_D3D12SDKConfiguration, 0x7cda6aca, 0xa03e, 0x49c8, 0x94, 0x58, 0x03, 0x34, 0xd2, 0x0e, 0x07, 0xce);`

They correspond, respectively, to the following interfaces.

- [ID3D12Debug interface](#)
- [ID3D12Tools interface](#)
- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [ID3D12SDKConfiguration interface](#)

`riid`

Type: `_In_ REFIID`

The globally unique identifier (**GUID**) for the SDK configuration interface. The **REFIID**, or **GUID**, of the interface can be obtained by using the `_uuidof` macro. For example, `_uuidof(ID3D12SDKConfiguration)` will retrieve the **GUID** of the debug interface.

`ppvDebug`

Type: `_COM_Outptr_opt_ void**`

The `out` parameter that contains the requested interface on return (for example, the SDK configuration interface), as a pointer to pointer to void. See [ID3D12SDKConfiguration](#).

Return value

Type: [HRESULT](#)

If the function succeeds, then it returns **S_OK**. Otherwise, it returns one of the [Direct3D 12 return codes](#).

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12Debug](#)

- ID3D12Tools
 - ID3D12DeviceRemovedExtendedDataSettings
 - ID3D12SDKConfiguration
-

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12SerializeRootSignature function (d3d12.h)

Article01/27/2022

Serializes a root signature version 1.0 that can be passed to [ID3D12Device::CreateRootSignature](#).

Syntax

C++

```
HRESULT D3D12SerializeRootSignature(
    [in]           const D3D12_ROOT_SIGNATURE_DESC *pRootSignature,
    [in]           D3D_ROOT_SIGNATURE_VERSION      Version,
    [out]          ID3DBlob                  **ppBlob,
    [out, optional] ID3DBlob                **ppErrorBlob
);
```

Parameters

[in] pRootSignature

Type: [const D3D12_ROOT_SIGNATURE_DESC*](#)

The description of the root signature, as a pointer to a [D3D12_ROOT_SIGNATURE_DESC](#) structure.

[in] Version

Type: [D3D_ROOT_SIGNATURE_VERSION](#)

A [D3D_ROOT_SIGNATURE_VERSION](#)-typed value that specifies the version of root signature.

[out] ppBlob

Type: [ID3DBlob**](#)

A pointer to a memory block that receives a pointer to the [ID3DBlob](#) interface that you can use to access the serialized root signature.

[out, optional] ppErrorBlob

Type: [ID3DBlob**](#)

A pointer to a memory block that receives a pointer to the [ID3DBlob](#) interface that you can use to access serializer error messages, or **NULL** if there are no errors.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

Remarks

This function has been superceded by [D3D12SerializeVersionedRootSignature](#) as of the Windows 10 Anniversary Update (14393).

If an application procedurally generates a [D3D12_ROOT_SIGNATURE_DESC](#) data structure, it must pass a pointer to this [D3D12_ROOT_SIGNATURE_DESC](#) in a call to [D3D12SerializeRootSignature](#) to make the serialized form. The application then passes the serialized form to which *ppBlob* points into [ID3D12Device::CreateRootSignature](#).

If a shader has been authored with a root signature in it, the compiled shader will contain a serialized root signature in it already. In this case, pass the compiled shader blob to [ID3D12Device::CreateRootSignature](#) to obtain the runtime root signature object.

The function signature [PFN_D3D12_SERIALIZE_ROOT_SIGNATURE](#) is provided as a [typedef](#), so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

Examples

Create an empty root signature.

C++

```
CD3DX12_ROOT_SIGNATURE_DESC rootSignatureDesc;
rootSignatureDesc.Init(0, nullptr, 0, nullptr,
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);

ComPtr<ID3DBlob> signature;
ComPtr<ID3DBlob> error;
ThrowIfFailed(D3D12SerializeRootSignature(&rootSignatureDesc,
D3D_ROOT_SIGNATURE_VERSION_1, &signature, &error));
ThrowIfFailed(m_device->CreateRootSignature(0, signature-
>GetBufferPointer(), signature->GetBufferSize()),
```

```
IID_PPV_ARGS(&m_rootSignature));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Creating a Root Signature](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12SerializeVersionedRootSignature function (d3d12.h)

Article01/27/2022

Serializes a root signature of any version that can be passed to [ID3D12Device::CreateRootSignature](#).

Syntax

C++

```
HRESULT D3D12SerializeVersionedRootSignature(
    [in]           const D3D12_VERSIONED_ROOT_SIGNATURE_DESC *pRootSignature,
    [out]          ID3DBlob                                **ppBlob,
    [out, optional] ID3DBlob                                **ppErrorBlob
);
```

Parameters

[in] pRootSignature

Type: [const D3D12_VERSIONED_ROOT_SIGNATURE_DESC*](#)

Specifies a [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#) that contains a description of any version of a root signature.

[out] ppBlob

Type: [ID3DBlob**](#)

A pointer to a memory block that receives a pointer to the [ID3DBlob](#) interface that you can use to access the serialized root signature.

[out, optional] ppErrorBlob

Type: [ID3DBlob**](#)

A pointer to a memory block that receives a pointer to the [ID3DBlob](#) interface that you can use to access serializer error messages, or [NULL](#) if there are no errors.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

Remarks

If an application procedurally generates a [D3D12_ROOT_SIGNATURE_DESC1](#) data structure, it must pass a pointer to this [D3D12_ROOT_SIGNATURE_DESC1](#) in a call to [D3D12SerializeVersionedRootSignature](#) to make the serialized form. The application then passes the serialized form to which *ppBlob* points into [ID3D12Device::CreateRootSignature](#).

If a shader has been authored with a root signature in it, the compiled shader will contain a serialized root signature in it already. In this case, pass the compiled shader blob to [ID3D12Device::CreateRootSignature](#) to obtain the runtime root signature object.

Note that for Xbox developers, use of HLSL-authored root signatures is strongly recommended.

The function signature `PFN_D3D12_SERIALIZE_VERSIONED_ROOT_SIGNATURE` is provided as a `typedef`, so that you can use dynamic linking techniques ([GetProcAddress](#)) instead of statically linking.

This function was released with the Windows 10 Anniversary Update (14393) and supersedes [D3D12SerializeRootSignature](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

Core Functions

Creating a Root Signature

[D3DX12SerializeVersionedRootSignature](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandAllocator interface (d3d12.h)

Article 02/22/2024

Represents the allocations of storage for graphics processing unit (GPU) commands.

Inheritance

The **ID3D12CommandAllocator** interface inherits from [ID3D12Pageable](#).

ID3D12CommandAllocator also has these types of members:

Methods

The **ID3D12CommandAllocator** interface has these methods.

[] [Expand table](#)

[ID3D12CommandAllocator::Reset](#)

Indicates to re-use the memory that is associated with the command allocator.

Remarks

Use [ID3D12Device::CreateCommandAllocator](#) to create a command allocator object.

The command allocator object corresponds to the underlying allocations in which GPU commands are stored. The command allocator object applies to both direct command lists and bundles. You must use a command allocator object in a DirectX 12 app.

Examples

The [D3D12nBodyGravity](#) sample uses **ID3D12CommandAllocator** as follows:

Header file declarations.

C++

```

D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr m_swapChain;
ComPtr m_device;
ComPtr m_renderTargets[FrameCount];
ComPtr m_commandAllocator;
ComPtr m_commandQueue;
ComPtr m_rootSignature;
ComPtr m_rtvHeap;
ComPtr m_pipelineState;
ComPtr m_commandList;
UINT m_rtvDescriptorSize;

```

Asynchronous compute thread.

C++

```

DWORD D3D12nBodyGravity::AsyncComputeThreadProc(int threadIndex)
{
    ID3D12CommandQueue* pCommandQueue =
m_computeCommandQueue[threadIndex].Get();
    ID3D12CommandAllocator* pCommandAllocator =
m_computeAllocator[threadIndex].Get();
    ID3D12GraphicsCommandList* pCommandList =
m_computeCommandList[threadIndex].Get();
    ID3D12Fence* pFence = m_threadFences[threadIndex].Get();

    while (0 == InterlockedGetValue(&m_terminating))
    {
        // Run the particle simulation.
        Simulate(threadIndex);

        // Close and execute the command list.
        ThrowIfFailed(pCommandList->Close());
        ID3D12CommandList* ppCommandLists[] = { pCommandList };

        pCommandQueue->ExecuteCommandLists(1, ppCommandLists);

        // Wait for the compute shader to complete the simulation.
        UINT64 threadFenceValue =
InterlockedIncrement(&m_threadFenceValues[threadIndex]);
        ThrowIfFailed(pCommandQueue->Signal(pFence, threadFenceValue));
        ThrowIfFailed(pFence->SetEventOnCompletion(threadFenceValue,
m_threadFenceEvents[threadIndex]));
        WaitForSingleObject(m_threadFenceEvents[threadIndex], INFINITE);

        // Wait for the render thread to be done with the SRV so that
        // the next frame in the simulation can run.
        UINT64 renderContextFenceValue =
InterlockedGetValue(&m_renderContextFenceValues[threadIndex]);
        if (m_renderContextFence->GetCompletedValue() <

```

```

renderContextFenceValue)
{
    ThrowIfFailed(pCommandQueue->Wait(m_renderContextFence.Get(),
renderContextFenceValue));
    InterlockedExchange(&m_renderContextFenceValues[threadIndex],
0);
}

// Swap the indices to the SRV and UAV.
m_srvIndex[threadIndex] = 1 - m_srvIndex[threadIndex];

// Prepare for the next frame.
ThrowIfFailed(pCommandAllocator->Reset());
ThrowIfFailed(pCommandList->Reset(pCommandAllocator,
m_computeState.Get()));
}

return 0;
}

```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandAllocator::Reset method (d3d12.h)

Article 06/26/2024

Indicates to re-use the memory that is associated with the command allocator.

Syntax

C++

```
HRESULT Reset();
```

Return value

Type: [HRESULT](#)

This method returns [E_FAIL](#) if there is an actively recording command list referencing the command allocator. The debug layer will also issue an error in this case.

See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

Your app calls **Reset** to re-use the memory that is associated with a command allocator. From this call to **Reset**, the runtime and driver assume that the graphics processing unit (GPU) is no longer executing any command lists that have recorded commands with the command allocator. So you should ensure that you don't call **Reset** until the GPU is done executing command lists associated with the allocator.

It's undefined behavior to call **Reset** on a command allocator while it has a command list still being executed.

The debug layer will issue a warning if it can't prove that there are no pending GPU references to command lists that have recorded commands in the allocator.

The debug layer will issue an error if **Reset** is called concurrently by multiple threads (on the same allocator object).

Examples

The D3D12HelloTriangle sample uses `ID3D12CommandAllocator::Reset` as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
// Command list allocators can only be reset when the associated
// command lists have finished execution on the GPU; apps should use
// fences to determine GPU execution progress.
ThrowIfFailed(m_commandAllocator->Reset());

// However, when ExecuteCommandList() is called on a particular command
// list, that command list can then be reset at any time and must be before
// re-recording.
ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
m_pipelineState.Get()));

// Set necessary state.
m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
m_commandList->RSSetViewports(1, &m_viewport);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
```

```
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),  
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));
```

```
ThrowIfFailed(m_commandList->Close());
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandAllocator](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandList interface (d3d12.h)

Article 02/22/2024

An interface from which [ID3D12GraphicsCommandList](#) inherits. It represents an ordered set of commands that the GPU executes, while allowing for extension to support other command lists than just those for graphics (such as compute and copy).

Inheritance

The [ID3D12CommandList](#) interface inherits from [ID3D12DeviceChild](#).

[ID3D12CommandList](#) also has these types of members:

Methods

The [ID3D12CommandList](#) interface has these methods.

 [Expand table](#)

ID3D12CommandList::GetType	Gets the type of the command list, such as direct, bundle, compute, or copy.

Remarks

Use [ID3D12Device::CreateCommandList](#) to create a command list object.

See also [ID3D12GraphicsCommandList](#), which derives from [ID3D12CommandList](#).

A command list corresponds to a set of commands that the graphics processing unit (GPU) executes. Commands set state, draw, clear, copy, and so on.

Direct3D 12 command lists only support these 2 levels of indirection:

- A direct command list corresponds to a command buffer that the GPU can execute.
- A bundle can be executed only directly via a direct command list.

Examples

The [D3D12nBodyGravity](#) sample uses [ID3D12CommandList](#) as follows:

C++

```
DWORD D3D12nBodyGravity::AsyncComputeThreadProc(int threadIndex)
{
    ID3D12CommandQueue* pCommandQueue =
m_computeCommandQueue[threadIndex].Get();
    ID3D12CommandAllocator* pCommandAllocator =
m_computeAllocator[threadIndex].Get();
    ID3D12GraphicsCommandList* pCommandList =
m_computeCommandList[threadIndex].Get();
    ID3D12Fence* pFence = m_threadFences[threadIndex].Get();

    while (0 == InterlockedGetValue(&m_terminating))
    {
        // Run the particle simulation.
        Simulate(threadIndex);

        // Close and execute the command list.
        ThrowIfFailed(pCommandList->Close());
        ID3D12CommandList* ppCommandLists[] = { pCommandList };

        pCommandQueue->ExecuteCommandLists(1, ppCommandLists);

        // Wait for the compute shader to complete the simulation.
        UINT64 threadFenceValue =
InterlockedIncrement(&m_threadFenceValues[threadIndex]);
        ThrowIfFailed(pCommandQueue->Signal(pFence, threadFenceValue));
        ThrowIfFailed(pFence->SetEventOnCompletion(threadFenceValue,
m_threadFenceEvents[threadIndex]));
        WaitForSingleObject(m_threadFenceEvents[threadIndex], INFINITE);

        // Wait for the render thread to be done with the SRV so that
        // the next frame in the simulation can run.
        UINT64 renderContextFenceValue =
InterlockedGetValue(&m_renderContextFenceValues[threadIndex]);
        if (m_renderContextFence->GetCompletedValue() <
renderContextFenceValue)
        {
            ThrowIfFailed(pCommandQueue->Wait(m_renderContextFence.Get(),
renderContextFenceValue));
            InterlockedExchange(&m_renderContextFenceValues[threadIndex],
0);
        }

        // Swap the indices to the SRV and UAV.
        m_srvIndex[threadIndex] = 1 - m_srvIndex[threadIndex];

        // Prepare for the next frame.
        ThrowIfFailed(pCommandAllocator->Reset());
        ThrowIfFailed(pCommandList->Reset(pCommandAllocator,
m_computeState.Get()));
    }

    return 0;
}
```

```
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12DeviceChild](#)

[ID3D12GraphicsCommandList](#)

[Setting descriptor heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandList::GetType method (d3d12.h)

Article 02/22/2024

Gets the type of the command list, such as direct, bundle, compute, or copy.

Syntax

C++

```
D3D12_COMMAND_LIST_TYPE GetType();
```

Return value

Type: [D3D12_COMMAND_LIST_TYPE](#)

This method returns the type of the command list, as a [D3D12_COMMAND_LIST_TYPE](#) enumeration constant, such as direct, bundle, compute, or copy.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue interface (d3d12.h)

Article 07/27/2022

Provides methods for submitting command lists, synchronizing command list execution, instrumenting the command queue, and updating resource tile mappings.

Inheritance

The **ID3D12CommandQueue** interface inherits from [ID3D12Pageable](#).

ID3D12CommandQueue also has these types of members:

Methods

The **ID3D12CommandQueue** interface has these methods.

[+] [Expand table](#)

ID3D12CommandQueue::BeginEvent
Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (ID3D12CommandQueue.BeginEvent)
ID3D12CommandQueue::CopyTileMappings
Copies mappings from a source reserved resource to a destination reserved resource.
ID3D12CommandQueue::EndEvent
Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (ID3D12CommandQueue.EndEvent)
ID3D12CommandQueue::ExecuteCommandLists
Submits an array of command lists for execution.
ID3D12CommandQueue::GetClockCalibration
This method samples the CPU and GPU timestamp counters at the same moment in time.
ID3D12CommandQueue::GetDesc

Gets the description of the command queue.
ID3D12CommandQueue::GetTimestampFrequency
This method is used to determine the rate at which the GPU timestamp counter increments.
ID3D12CommandQueue::SetMarker
Not intended to be called directly. Use the PIX event runtime to insert events into a command queue. (ID3D12CommandQueue::SetMarker)
ID3D12CommandQueue::Signal
Updates a fence to a specified value.
ID3D12CommandQueue::UpdateTileMappings
Updates mappings of tile locations in reserved resources to memory locations in a resource heap.
ID3D12CommandQueue::Wait
Queues a GPU-side wait, and returns immediately. A GPU-side wait is where the GPU waits until the specified fence reaches or exceeds the specified value.

Remarks

Use [ID3D12Device::CreateCommandQueue](#) to create a command queue object.

Examples

The [D3D12nBodyGravity](#) sample uses **ID3D12CommandQueue** as follows:

Header file declarations.

C++

```
// Compute objects.
ComPtr<ID3D12CommandAllocator> m_computeAllocator[ThreadCount];
ComPtr<ID3D12CommandQueue> m_computeCommandQueue[ThreadCount];
ComPtr<ID3D12GraphicsCommandList> m_computeCommandList[ThreadCount];
```

Asynchronous compute thread.

C++

```

DWORD D3D12nBodyGravity::AsyncComputeThreadProc(int threadIndex)
{
    ID3D12CommandQueue* pCommandQueue =
m_computeCommandQueue[threadIndex].Get();
    ID3D12CommandAllocator* pCommandAllocator =
m_computeAllocator[threadIndex].Get();
    ID3D12GraphicsCommandList* pCommandList =
m_computeCommandList[threadIndex].Get();
    ID3D12Fence* pFence = m_threadFences[threadIndex].Get();

    while (0 == InterlockedGetValue(&m_terminating))
    {
        // Run the particle simulation.
        Simulate(threadIndex);

        // Close and execute the command list.
        ThrowIfFailed(pCommandList->Close());
        ID3D12CommandList* ppCommandLists[] = { pCommandList };

        pCommandQueue->ExecuteCommandLists(1, ppCommandLists);

        // Wait for the compute shader to complete the simulation.
        UINT64 threadFenceValue =
InterlockedIncrement(&m_threadFenceValues[threadIndex]);
        ThrowIfFailed(pCommandQueue->Signal(pFence, threadFenceValue));
        ThrowIfFailed(pFence->SetEventOnCompletion(threadFenceValue,
m_threadFenceEvents[threadIndex]));
        WaitForSingleObject(m_threadFenceEvents[threadIndex], INFINITE);

        // Wait for the render thread to be done with the SRV so that
        // the next frame in the simulation can run.
        UINT64 renderContextFenceValue =
InterlockedGetValue(&m_renderContextFenceValues[threadIndex]);
        if (m_renderContextFence->GetCompletedValue() <
renderContextFenceValue)
        {
            ThrowIfFailed(pCommandQueue->Wait(m_renderContextFence.Get(),
renderContextFenceValue));
            InterlockedExchange(&m_renderContextFenceValues[threadIndex],
0);
        }

        // Swap the indices to the SRV and UAV.
        m_srvIndex[threadIndex] = 1 - m_srvIndex[threadIndex];

        // Prepare for the next frame.
        ThrowIfFailed(pCommandAllocator->Reset());
        ThrowIfFailed(pCommandList->Reset(pCommandAllocator,
m_computeState.Get()));
    }

    return 0;
}

```

}

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::BeginEvent method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command queue.

Syntax

C++

```
void BeginEvent(
    UINT      Metadata,
    [in, optional] const void *pData,
    UINT      Size
);
```

Parameters

Metadata

Type: [UINT](#)

Internal.

[in, optional] pData

Type: [const void*](#)

Internal.

Size

Type: [UINT](#)

Internal.

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To mark the start of an instrumentation region at the current location within a D3D12 command queue, use the **PIXBeginEvent** function or **PIXScopedEvent** macro. These are provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::CopyTileMappings method (d3d12.h)

Article 02/22/2024

Copies mappings from a source reserved resource to a destination reserved resource.

Syntax

C++

```
void CopyTileMappings(
    [in] ID3D12Resource                  *pDstResource,
    [in] const D3D12_TILED_RESOURCE_COORDINATE *pDstRegionStartCoordinate,
    [in] ID3D12Resource                  *pSrcResource,
    [in] const D3D12_TILED_RESOURCE_COORDINATE *pSrcRegionStartCoordinate,
    [in] const D3D12_TILE_REGION_SIZE        *pRegionSize,
    D3D12_TILE_MAPPING_FLAGS              Flags
);
```

Parameters

[in] pDstResource

A pointer to the destination reserved resource.

[in] pDstRegionStartCoordinate

A pointer to a [D3D12_TILED_RESOURCE_COORDINATE](#) structure that describes the starting coordinates of the destination reserved resource.

[in] pSrcResource

A pointer to the source reserved resource.

[in] pSrcRegionStartCoordinate

A pointer to a [D3D12_TILED_RESOURCE_COORDINATE](#) structure that describes the starting coordinates of the source reserved resource.

[in] pRegionSize

A pointer to a [D3D12_TILE_REGION_SIZE](#) structure that describes the size of the reserved region.

Flags

One member of [D3D12_TILE_MAPPING_FLAGS](#).

Return value

None

Remarks

Use **CopyTileMappings** to copy the tile mappings from one reserved resource to another, either to duplicate a resource mapping, or to initialize a new mapping before modifying it using [UpdateTileMappings](#).

CopyTileMappings helps with tasks such as shifting mappings around within and across reserved resources, for example, scrolling tiles. The source and destination regions can overlap; the result of the copy in this situation is as if the source was saved to a temporary location and from there written to the destination.

The destination and the source regions must each entirely fit in their resource or behavior is undefined and the debug layer will emit an error.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

[UpdateTileMappings](#)

[Volume Tiled Resources](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::EndEvent method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command queue.

Syntax

C++

```
void EndEvent();
```

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To mark the end of an instrumentation region at the current location within a D3D12 command queue, use the **PIXEndEvent** function or **PIXScopedEvent** macro. These are provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::ExecuteCommandLists method (d3d12.h)

Article 10/13/2021

Submits an array of command lists for execution.

Syntax

C++

```
void ExecuteCommandLists(
    [in] UINT             NumCommandLists,
    [in] ID3D12CommandList * const *ppCommandLists
);
```

Parameters

[in] NumCommandLists

The number of command lists to be executed.

[in] ppCommandLists

The array of [ID3D12CommandList](#) command lists to be executed.

Return value

None

Remarks

Calling **ExecuteCommandLists** twice in succession (from the same thread, or different threads) guarantees that the first workload (A) finishes before the second workload (B). Calling **ExecuteCommandLists** with two command lists allows the driver to merge the two command lists such that the second command list (D) may begin executing work before all work from the first (C) has finished. Specifically, your application is allowed to insert a fence signal or wait between A and B, and the driver has no visibility into this, so the driver must ensure that everything in A is complete before the fence operation.

There is no such opportunity in a single call to the API, so the driver is able to optimize that scenario.

The driver is free to patch the submitted command lists. It is the calling application's responsibility to ensure that the graphics processing unit (GPU) is not currently reading the any of the submitted command lists from a previous execution.

Applications are encouraged to batch together command list executions to reduce fixed costs associated with submitted commands to the GPU.

Runtime validation

Bundles can't be submitted to a command queue directly. If a bundle is passed to this method, the runtime will drop the call. The runtime will also drop the call if the [Close](#) function has not been called on one or more of the command lists.

The runtime will detect if the command allocators associated with the command lists have been reset after [Close](#) was called. The runtime will drop the call and remove the device in this situation.

The runtime will drop the call and remove the device if the command queue fence indicates that a previous execution of any of the command lists has not yet completed.

The runtime will validate the "before" and "after" states of resource transition barriers inside of [ExecuteCommandLists](#). If the "before" state of a transition does not match up with the "after" state of a previous transition, then the runtime will drop the call and remove the device.

The runtime will validate the "before" and "after" states of queries used by the command lists. If an error is detected, then the runtime will drop the call and remove the device.

Debug layer

The debug layer issues errors for all cases where the runtime would drop the call.

The debug layer issues an error if it detects that any resource referenced by the command lists, including queries, has been destroyed.

Examples

Renders a scene.

```
// Pipeline objects.  
D3D12_VIEWPORT m_viewport;  
ComPtr m_swapChain;  
ComPtr m_d3d11DeviceContext;  
ComPtr m_d3d11On12Device;  
ComPtr m_d3d12Device;  
ComPtr m_dWriteFactory;  
ComPtr m_d2dFactory;  
ComPtr m_d2dDevice;  
ComPtr m_d2dDeviceContext;  
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];  
ComPtr<ID3D11Resource> m_wrappedBackBuffers[FrameCount];  
ComPtr<ID2D1Bitmap1> m_d2dRenderTargets[FrameCount];  
ComPtr<ID3D12CommandAllocator> m_commandAllocators[FrameCount];  
ComPtr<ID3D12CommandQueue> m_commandQueue;  
ComPtr<ID3D12RootSignature> m_rootSignature;  
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;  
ComPtr<ID3D12PipelineState> m_pipelineState;  
ComPtr<ID3D12GraphicsCommandList> m_commandList;  
D3D12_RECT m_scissorRect;
```

C++

```
// Render the scene.  
void D3D11On12::OnRender()  
{  
    // Record all the commands we need to render the scene into the command  
    // list.  
    PopulateCommandList();  
  
    // Execute the command list.  
    ID3D12CommandList* ppCommandLists[] = { m_commandList.Get() };  
    m_commandQueue->ExecuteCommandLists(_countof(ppCommandLists),  
    ppCommandLists);  
  
    RenderUI();  
  
    // Present the frame.  
    ThrowIfFailed(m_swapChain->Present(1, 0));  
  
    MoveToNextFrame();  
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::GetClockCalibration method (d3d12.h)

Article 02/22/2024

This method samples the CPU and GPU timestamp counters at the same moment in time.

Syntax

C++

```
HRESULT GetClockCalibration(  
    [out] UINT64 *pGpuTimestamp,  
    [out] UINT64 *pCpuTimestamp  
);
```

Parameters

[out] pGpuTimestamp

Type: **UINT64***

The value of the GPU timestamp counter.

[out] pCpuTimestamp

Type: **UINT64***

The value of the CPU timestamp counter.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

For more information, refer to [Timing](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12CommandQueue::GetDesc method (d3d12.h)

Article 02/22/2024

Gets the description of the command queue.

Syntax

C++

```
D3D12_COMMAND_QUEUE_DESC GetDesc();
```

Return value

Type: [D3D12_COMMAND_QUEUE_DESC](#)

The description of the command queue, as a [D3D12_COMMAND_QUEUE_DESC](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::GetTimestampFrequency method (d3d12.h)

Article02/22/2024

This method is used to determine the rate at which the GPU timestamp counter increments.

Syntax

C++

```
HRESULT GetTimestampFrequency(  
    [out] UINT64 *pFrequency  
);
```

Parameters

[out] pFrequency

Type: **UINT64***

The GPU timestamp counter frequency (in ticks/second).

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

For more information, refer to [Timing](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::SetMarker method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command queue.

Syntax

C++

```
void SetMarker(
    UINT      Metadata,
    [in, optional] const void *pData,
    UINT      Size
);
```

Parameters

Metadata

Type: **UINT**

Internal.

[in, optional] pData

Type: **const void***

Internal.

Size

Type: **UINT**

Internal.

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To insert instrumentation markers at the current location within a D3D12 command queue, use the **PIXSetMarker** function. This is provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::Signal method (d3d12.h)

Article 02/22/2024

Updates a fence to a specified value.

Syntax

C++

```
HRESULT Signal(  
    ID3D12Fence *pFence,  
    UINT64      Value  
>;
```

Parameters

pFence

Type: [ID3D12Fence*](#)

A pointer to the [ID3D12Fence](#) object.

Value

Type: [UINT64](#)

The value to set the fence to.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Use this method to set a fence value from the GPU side. Use [ID3D12Fence::Signal](#) to set a fence from the CPU side.

Examples

Adds a signal to the command queue, then waits for the compute shader to complete the simulation, finally signal and increment the fence value.

C++

```
// Wait for the compute shader to complete the simulation.  
UINT64 threadFenceValue =  
InterlockedIncrement(&m_threadFenceValues[threadIndex]);  
ThrowIfFailed(pCommandQueue->Signal(pFence, threadFenceValue));  
ThrowIfFailed(pFence->SetEventOnCompletion(threadFenceValue,  
m_threadFenceEvents[threadIndex]));  
WaitForSingleObject(m_threadFenceEvents[threadIndex], INFINITE);
```

C++

```
// Add a signal command to the queue.  
ThrowIfFailed(m_commandQueue->Signal(m_renderContextFence.Get(),  
m_renderContextFenceValue));
```

C++

```
// Signal and increment the fence value.  
ThrowIfFailed(m_commandQueue->Signal(m_renderContextFence.Get(),  
m_renderContextFenceValue));  
m_renderContextFenceValue++;
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandQueue](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::UpdateTileMap pings method (d3d12.h)

Article 10/13/2021

Updates mappings of tile locations in reserved resources to memory locations in a resource heap.

Syntax

C++

```
void UpdateTileMappings(
    [in]          ID3D12Resource                  *pResource,
                  UINT                         NumResourceRegions,
    [in, optional] const D3D12_TILED_RESOURCE_COORDINATE
    *pResourceRegionStartCoordinates,
    [in, optional] const D3D12_TILE_REGION_SIZE
    *pResourceRegionSizes,
    [in, optional] ID3D12Heap                     *pHeap,
                  UINT                         NumRanges,
    [in, optional] const D3D12_TILE_RANGE_FLAGS
    [in, optional] const UINT
    *pHeapRangeStartOffsets,
    [in, optional] const UINT
                  D3D12_TILE_MAPPING_FLAGS
    *pRangeTileCounts,
    Flags
);
```

Parameters

[in] pResource

A pointer to the reserved resource.

NumResourceRegions

The number of reserved resource regions.

[in, optional] pResourceRegionStartCoordinates

An array of [D3D12_TILED_RESOURCE_COORDINATE](#) structures that describe the starting coordinates of the reserved resource regions. The *NumResourceRegions* parameter specifies the number of [D3D12_TILED_RESOURCE_COORDINATE](#) structures in the array.

[in, optional] pResourceRegionSizes

An array of [D3D12_TILE_REGION_SIZE](#) structures that describe the sizes of the reserved resource regions. The *NumResourceRegions* parameter specifies the number of [D3D12_TILE_REGION_SIZE](#) structures in the array.

[in, optional] pHheap

A pointer to the resource heap.

NumRanges

The number of tile ranges.

[in, optional] pRangeFlags

A pointer to an array of [D3D12_TILE_RANGE_FLAGS](#) values that describes each tile range. The *NumRanges* parameter specifies the number of values in the array.

[in, optional] pHheapRangeStartOffsets

An array of offsets into the resource heap. These are 0-based tile offsets, counting in tiles (not bytes).

[in, optional] pRangeTileCounts

An array of tiles. An array of values that specify the number of tiles in each tile range. The *NumRanges* parameter specifies the number of values in the array.

Flags

A combination of [D3D12_TILE_MAPPING_FLAGS](#) values that are combined by using a bitwise OR operation.

Return value

None

Remarks

Use [UpdateTileMappings](#) to map the virtual pages of a reserved resource to the physical pages of a heap. The mapping does not have to be in order. The operation is similar to [ID3D11DeviceContext2::UpdateTileMappings](#) with the one key difference that D3D12 allows a reserved resource to have tiles from multiple heaps.

In a single call to **UpdateTileMappings**, you can map one or more ranges of resource tiles to one or more ranges of heap tiles.

You can organize the parameters of **UpdateTileMappings** in these ways to perform an update:

- **Reserved resource whose mappings are updated.** Mappings start off all NULL when a resource is initially created.
- **Set of tile regions on the reserved resource whose mappings are updated.** You can make one **UpdateTileMappings** call to update many mappings or multiple calls with a bit more API call overhead as well if that is more convenient.
 - *NumResourceRegions* specifies how many regions there are.
 - *pResourceRegionStartCoordinates* and *pResourceRegionSizes* are each arrays that identify the start location and extend of each region. If *NumResourceRegions* is 1, for convenience either or both of the arrays that describe the regions can be NULL. NULL for *pResourceRegionStartCoordinates* means the start coordinate is all 0s, and NULL for *pResourceRegionSizes* identifies a default region that is the full set of tiles for the entire reserved resource, including all mipmaps, array slices, or both.
 - If *pResourceRegionStartCoordinates* isn't NULL and *pResourceRegionSizes* is NULL, the region size defaults to 1 tile for all regions. This makes it easy to define mappings for a set of individual tiles each at disparate locations by providing an array of locations in *pResourceRegionStartCoordinates* without having to send an array of *pResourceRegionSizes* all set to 1.

The updates are applied from first region to last; so, if regions overlap in a single call, the updates later in the list overwrite the areas that overlap with previous updates.

- **Heap that provides memory where tile mappings can go.** If **UpdateTileMappings** only defines NULL mappings, you don't need to specify a heap.
- **Set of tile ranges where mappings are going.** Each given tile range can specify one of a few types of ranges: a range of tiles in a heap (default), a count of tiles in the reserved resource to map to a single tile in a heap (sharing the tile), a count of tile mappings in the reserved resource to skip and leave as they are, or a count of tiles in the heap to map to NULL. *NumRanges* specifies the number of tile ranges, where the total tiles identified across all ranges must match the total number of tiles in the tile regions from the previously described reserved resource. Mappings are defined by iterating through the tiles in the tile regions in sequential order - x then y then z order for box regions - while walking through the set of tile ranges in sequential order. The breakdown of tile regions doesn't have to line up with the breakdown of tile ranges, but the total number of tiles on both sides must be equal so that each reserved resource tile specified has a mapping specified.

pRangeFlags, *pHeapRangeStartOffsets*, and *pRangeTileCounts* are all arrays, of size *NumRanges*, that describe the tile ranges. If *pRangeFlags* is NULL, all ranges are sequential tiles in the heap; otherwise, for each range *i*, *pRangeFlags[i]* identifies how the mappings in that range of tiles work:

- If *pRangeFlags[i]* is **D3D12_TILE_RANGE_FLAG_NONE**, that range defines sequential tiles in the heap, with the number of tiles being *pRangeTileCounts[i]* and the starting location *pHeapRangeStartOffsets[i]*. If *NumRanges* is 1, *pRangeTileCounts* can be NULL and defaults to the total number of tiles specified by all the tile regions.
- If *pRangeFlags[i]* is **D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE**, *pHeapRangeStartOffsets[i]* identifies the single tile in the heap to map to, and *pRangeTileCounts[i]* specifies how many tiles from the tile regions to map to that heap location. If *NumRanges* is 1, *pRangeTileCounts* can be NULL and defaults to the total number of tiles specified by all the tile regions.
- If *pRangeFlags[i]* is **D3D12_TILE_RANGE_FLAG_NULL**, *pRangeTileCounts[i]* specifies how many tiles from the tile regions to map to NULL. If *NumRanges* is 1, *pRangeTileCounts* can be NULL and defaults to the total number of tiles specified by all the tile regions. *pHeapRangeStartOffsets[i]* is ignored for NULL mappings.
- If *pRangeFlags[i]* is **D3D12_TILE_RANGE_FLAG_SKIP**, *pRangeTileCounts[i]* specifies how many tiles from the tile regions to skip over and leave existing mappings unchanged for. This can be useful if a tile region conveniently bounds an area of tile mappings to update except with some exceptions that need to be left the same as whatever they were mapped to before. *pHeapRangeStartOffsets[i]* is ignored for SKIP mappings.

Reserved resources must follow the same rules for tile aliasing, initialization, and data inheritance as placed resources. See [CreatePlacedResource](#) for more details.

Here are some examples of common [UpdateTileMappings](#) cases:

Examples

The examples reference the following structures and enums:

- [D3D12_TILED_RESOURCE_COORDINATE](#)
- [D3D12_TILE_REGION_SIZE](#)
- [D3D12_TILE_RANGE_FLAGS](#)

Clearing an entire surface's mappings to NULL

C++

```

// - NULL for pResourceRegionStartCoordinates and pResourceRegionSizes
// defaults to the entire resource
// - NULL for pHheapRangeStartOffsets since it isn't needed for mapping tiles
// to NULL
// - NULL for pRangeTileCounts when NumRanges is 1 defaults to the same
// number of tiles as the resource region (which is
// the entire surface in this case)
//
UINT RangeFlags = D3D12_TILE_RANGE_FLAG_NULL;
pCommandQueue->UpdateTileMappings(pResource, 1, NULL, NULL, NULL, 1,
&RangeFlags, NULL, NULL, D3D12_TILE_MAPPING_FLAG_NONE);

```

Mapping a region of tiles to a single tile:

C++

```

// - This maps a 2x3 tile region at tile offset (1,1) in a resource to tile
[12] in a heap
//
D3D12_TILED_RESOURCE_COORDINATE TRC;
TRC.X = 1;
TRC.Y = 1;
TRC.Z = 0;
TRC.Subresource = 0;

D3D12_TILE_REGION_SIZE TRS;
TRS.bUseBox = TRUE;
TRS.Width = 2;
TRS.Height = 3;
TRS.Depth = 1;
TRS.NumTiles = TRS.Width * TRS.Height * TRS.Depth;

UINT RangeFlags = D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE;
UINT StartOffset = 12;

pCommandQueue-
>UpdateTileMappings(pResource,1,&TRC,&TRS,pHeap,1,&RangeFlags,&StartOffset,N
ULL,D3D12_TILE_MAPPING_FLAG_NONE);

```

Defining mappings for a set of disjoint individual tiles:

C++

```

// - This can also be accomplished in multiple calls.
// A single call to define multiple mapping updates can reduce CPU call
overhead slightly,
// at the cost of having to pass arrays as parameters.
// - Passing NULL for pResourceRegionSizes defaults to each region in the
resource

```

```

// being a single tile. So all that is needed are the coordinates of each
one.
// - Passing NULL for pRangeFlags defaults to no flags (since none are
needed in this case)
// - Passing NULL for pRangeTileCounts defaults to each range in the heap
being size 1.
// So all that is needed are the start offsets for each tile in the heap
//
D3D12_TILED_RESOURCE_COORDINATE TRC[3];
UINT StartOffsets[3];
UINT NumSingleTiles = 3;
TRC[0].X = 1;
TRC[0].Y = 1;
TRC[0].Subresource = 0;

StartOffsets[0] = 1;
TRC[1].X = 4;
TRC[1].Y = 7;
TRC[1].Subresource = 0;
StartOffsets[1] = 4;

TRC[2].X = 2;
TRC[2].Y = 3;
TRC[2].Subresource = 0;
StartOffsets[2] = 7;

pCommandQueue-
>UpdateTileMappings(pResource,NumSingleTiles,&TRC,NULL,pHeap,NumSingleTiles,
NULL,StartOffsets,
NULL,D3D12_TILE_MAPPING_FLAG_NONE);

```

Complex example - defining mappings for regions with some skips, some NULL mappings.

C++

```

// - This complex example hard codes the parameter arrays, whereas in
practice the
// application would likely configure the parameters programatically or in
a data driven way.
// - Suppose we have 3 regions in a resource to configure mappings for, 2x3
at coordinate (1,1),
// 3x3 at coordinate (4,7), and 7x1 at coordinate (20,30)
// - The tiles in the regions are walked from first to last, in X then Y
then Z order,
// while stepping forward through the specified Tile Ranges to determine
each mapping.
// In this example, 22 tile mappings need to be defined.
// - Suppose we want the first 3 tiles to be mapped to a contiguous range in
the heap starting at
// heap location [9], the next 8 to be skipped (left unchanged), the next
2 to map to NULL,

```

```
// the next 5 to share a single tile (heap location [17]) and the
// remaining
// 4 tiles to each map to unique heap locations, [2], [9], [4] and
// [17]:
//
D3D12_TILED_RESOURCE_COORDINATE TRC[3];
D3D12_TILE_REGION_SIZE TRS[3];
UINT NumRegions = 3;

TRC[0].X = 1;
TRC[0].Y = 1;
TRC[0].Subresource = 0;
TRS[0].bUseBox = TRUE;
TRS[0].Width = 2;
TRS[0].Height = 3;
TRS[0].NumTiles = TRS[0].Width * TRS[0].Height;

TRC[1].X = 4;
TRC[1].Y = 7;
TRC[1].Subresource = 0;
TRS[1].bUseBox = TRUE;
TRS[1].Width = 3;
TRS[1].Height = 3;
TRS[1].NumTiles = TRS[1].Width * TRS[1].Height;

TRC[2].X = 20;
TRC[2].Y = 30;
TRC[2].Subresource = 0;
TRS[2].bUseBox = TRUE;
TRS[2].Width = 7;
TRS[2].Height = 1;
TRS[2].NumTiles = TRS[2].Width * TRS[2].Height;

UINT NumRanges = 8;
UINT RangeFlags[8];
UINT HeapRangeStartOffsets[8];
UINT RangeTileCounts[8];

RangeFlags[0] = 0;
HeapRangeStartOffsets[0] = 9;
RangeTileCounts[0] = 3;

RangeFlags[1] = D3D12_TILE_RANGE_FLAG_SKIP;
HeapRangeStartOffsets[1] = 0; // offset is ignored for skip mappings
RangeTileCounts[1] = 8;

RangeFlags[2] = D3D12_TILE_RANGE_FLAG_NULL;
HeapRangeStartOffsets[2] = 0; // offset is ignored for NULL mappings
RangeTileCounts[2] = 2;

RangeFlags[3] = D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE;
HeapRangeStartOffsets[3] = 17;
RangeTileCounts[3] = 5;

RangeFlags[4] = 0;
```

```

HeapRangeStartOffsets[4] = 2;
RangeTileCounts[4] = 1;

RangeFlags[5] = 0;
HeapRangeStartOffsets[5] = 9;
RangeTileCounts[5] = 1;

RangeFlags[6] = 0;
HeapRangeStartOffsets[6] = 4;
RangeTileCounts[6] = 1;

RangeFlags[7] = 0;
HeapRangeStartOffsets[7] = 17;
RangeTileCounts[7] = 1;

pCommandQueue-
>UpdateTileMappings(pResource,NumRegions,TRC,TRS,pHeap,NumRanges,RangeFlags,
HeapRangeStartOffsets,RangeTileCounts,D3D12_TILE_MAPPING_FLAG_NONE);

```

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[CopyTileMappings](#)

[ID3D12CommandQueue](#)

[Volume Tiled Resources](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandQueue::Wait method (d3d12.h)

Article 02/22/2024

Queues a GPU-side wait, and returns immediately. A GPU-side wait is where the GPU waits until the specified fence reaches or exceeds the specified value.

Syntax

C++

```
HRESULT Wait(  
    ID3D12Fence *pFence,  
    UINT64      Value  
) ;
```

Parameters

pFence

Type: [ID3D12Fence*](#)

A pointer to the [ID3D12Fence](#) object.

Value

Type: [UINT64](#)

The value that the command queue is waiting for the fence to reach or exceed. So when [ID3D12Fence::GetCompletedValue](#) is greater than or equal to *Value*, the wait is terminated.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Because a wait is being queued, the API returns immediately. It's the command queue that waits (during which time no work is executed) until the fence specified reaches the requested value.

If you want to perform a CPU-side wait (where the calling thread blocks until a fence reaches a particular value), then you should use the [ID3D12Fence::SetEventOnCompletion](#) API in conjunction with [WaitForSingleObject](#) (or a similar API).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandQueue](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12CommandSignature interface (d3d12.h)

Article 02/22/2024

A command signature object enables apps to specify indirect drawing, including the buffer format, command type and resource bindings to be used.

Inheritance

The [ID3D12CommandSignature](#) interface inherits from the [ID3D12Pageable](#) interface.

Remarks

To create a command signature, call [ID3D12Device::CreateCommandSignature](#), as described in [Indirect Drawing](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ExecuteIndirect](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12DescriptorHeap interface (d3d12.h)

Article 02/22/2024

A descriptor heap is a collection of contiguous allocations of descriptors, one allocation for every descriptor. Descriptor heaps contain many object types that are not part of a Pipeline State Object (PSO), such as Shader Resource Views (SRVs), Unordered Access Views (UAVs), Constant Buffer Views (CBVs), and Samplers.

Inheritance

The **ID3D12DescriptorHeap** interface inherits from [ID3D12Pageable](#).

Methods

The **ID3D12DescriptorHeap** interface has these methods.

[+] Expand table

ID3D12DescriptorHeap::GetCPUDescriptorHandleForHeapStart
Gets the CPU descriptor handle that represents the start of the heap.
ID3D12DescriptorHeap::GetDesc
Gets the descriptor heap description.
ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart
Gets the GPU descriptor handle that represents the start of the heap.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DescriptorHeap::GetCPUDescriptorHandleForHeapStart method (d3d12.h)

Article02/22/2024

Gets the CPU descriptor handle that represents the start of the heap.

Syntax

C++

```
D3D12_CPU_DESCRIPTOR_HANDLE GetCPUDescriptorHandleForHeapStart();
```

Return value

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Returns the CPU descriptor handle that represents the start of the heap.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DescriptorHeap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DescriptorHeap::GetDesc method (d3d12.h)

Article 02/22/2024

Gets the descriptor heap description.

Syntax

C++

```
D3D12_DESCRIPTOR_HEAP_DESC GetDesc();
```

Return value

Type: [D3D12_DESCRIPTOR_HEAP_DESC](#)

The description of the descriptor heap, as a [D3D12_DESCRIPTOR_HEAP_DESC](#) structure.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12DescriptorHeap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart method (d3d12.h)

Article02/22/2024

Gets the GPU descriptor handle that represents the start of the heap.

Syntax

C++

```
D3D12_GPU_DESCRIPTOR_HANDLE GetGPUDescriptorHandleForHeapStart();
```

Return value

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

Returns the GPU descriptor handle that represents the start of the heap. If the descriptor heap is not shader-visible, a null handle is returned.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DescriptorHeap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device interface (d3d12.h)

Article06/12/2024

Represents a virtual adapter; it is used to create command allocators, command lists, command queues, fences, resources, pipeline state objects, heaps, root signatures, samplers, and many resource views.

Note This interface was introduced in Windows 10. Applications targetting Windows 10 should use this interface instead of later versions. Applications targetting a later version of Windows 10 should use the appropriate version of the **ID3D12Device** interface. The latest version of this interface is [ID3D12Device3](#) introduced in Windows 10 Fall Creators Update.

Inheritance

The **ID3D12Device** interface inherits from [ID3D12Object](#). **ID3D12Device** also has these types of members:

Methods

The **ID3D12Device** interface has these methods.

[+] Expand table

ID3D12Device::CheckFeatureSupport
Gets information about the features that are supported by the current graphics driver. (ID3D12Device.CheckFeatureSupport)
ID3D12Device::CopyDescriptors
Copies descriptors from a source to a destination. (ID3D12Device.CopyDescriptors)
ID3D12Device::CopyDescriptorsSimple
Copies descriptors from a source to a destination. (ID3D12Device.CopyDescriptorsSimple)
ID3D12Device::CreateCommandAllocator

Creates a command allocator object.
ID3D12Device::CreateCommandList
Creates a command list.
ID3D12Device::CreateCommandQueue
Creates a command queue.
ID3D12Device::CreateCommandSignature
This method creates a command signature.
ID3D12Device::CreateCommittedResource
Creates both a resource and an implicit heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.
ID3D12Device::CreateComputePipelineState
Creates a compute pipeline state object.
ID3D12Device::CreateConstantBufferView
Creates a constant-buffer view for accessing resource data.
ID3D12Device::CreateDepthStencilView
Creates a depth-stencil view for accessing resource data.
ID3D12Device::CreateDescriptorHeap
Creates a descriptor heap object.
ID3D12Device::CreateFence
Creates a fence object. (<code>ID3D12Device.CreateFence</code>)
ID3D12Device::CreateGraphicsPipelineState
Creates a graphics pipeline state object.
ID3D12Device::CreateHeap
Creates a heap that can be used with placed resources and reserved resources.
ID3D12Device::CreatePlacedResource

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

[ID3D12Device::CreateQueryHeap](#)

Creates a query heap. A query heap contains an array of queries.

[ID3D12Device::CreateRenderTargetView](#)

Creates a render-target view for accessing resource data. ([ID3D12Device.CreateRenderTargetView](#))

[ID3D12Device::CreateReservedResource](#)

Creates a resource that is reserved, and not yet mapped to any pages in a heap.

[ID3D12Device::CreateRootSignature](#)

Creates a root signature layout.

[ID3D12Device::CreateSampler](#)

Create a sampler object that encapsulates sampling information for a texture.

[ID3D12Device::CreateShaderResourceView](#)

Creates a shader-resource view for accessing data in a resource.
([ID3D12Device.CreateShaderResourceView](#))

[ID3D12Device::CreateSharedHandle](#)

Creates a shared handle to a heap, resource, or fence object.

[ID3D12Device::CreateUnorderedAccessView](#)

Creates a view for unordered accessing.

[ID3D12Device::Evict](#)

Enables the page-out of data, which precludes GPU access of that data.

[ID3D12Device::GetAdapterLuid](#)

Gets a locally unique identifier for the current device (adapter).

[ID3D12Device::GetCopyableFootprints](#)

Gets a resource layout that can be copied. Helps the app fill-in
D3D12_PLACED_SUBRESOURCE_FOOTPRINT and D3D12_SUBRESOURCE_FOOTPRINT when
suballocating space in upload heaps.

ID3D12Device::GetCustomHeapProperties	Divulges the equivalent custom heap properties that are used for non-custom heap types, based on the adapter's architectural properties.
ID3D12Device::GetDescriptorHandleIncrementSize	Gets the size of the handle increment for the given type of descriptor heap. This value is typically used to increment a handle into a descriptor array by the correct amount.
ID3D12Device::GetDeviceRemovedReason	Gets the reason that the device was removed.
ID3D12Device::GetNodeCount	Reports the number of physical adapters (nodes) that are associated with this device.
ID3D12Device::GetResourceAllocationInfo	Gets the size and alignment of memory required for a collection of resources on this adapter.
ID3D12Device::GetResourceTiling	Gets info about how a tiled resource is broken into tiles. (<code>ID3D12Device.GetResourceTiling</code>)
ID3D12Device::MakeResident	Makes objects resident for the device.
ID3D12Device::OpenSharedHandle	Opens a handle for shared resources, shared heaps, and shared fences, by using HANDLE and REFIID.
ID3D12Device::OpenSharedHandleByName	Opens a handle for shared resources, shared heaps, and shared fences, by using Name and Access.
ID3D12Device::SetStablePowerState	A development-time aid for certain types of profiling and experimental prototyping.

Remarks

Use [D3D12CreateDevice](#) to create a device.

For Windows 10 Anniversary some additional functionality is available through [ID3D12Device1](#).

Examples

The [D3D1211on12](#) sample uses **ID3D12Device** as follows:

Header file declarations.

C++

```
// Pipeline objects.  
D3D12_VIEWPORT m_viewport;  
ComPtr<IDXGISwapChain3> m_swapChain;  
ComPtr<ID3D12Device> m_device;  
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];  
ComPtr<ID3D12Resource> m_depthStencil;  
ComPtr<ID3D12CommandAllocator> m_commandAllocator;  
ComPtr<ID3D12GraphicsCommandList> m_commandList;  
ComPtr<ID3D12CommandQueue> m_commandQueue;  
ComPtr<ID3D12RootSignature> m_rootSignature;  
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;  
ComPtr<ID3D12DescriptorHeap> m_cbvSrvHeap;  
ComPtr<ID3D12DescriptorHeap> m_dsvHeap;  
ComPtr<ID3D12DescriptorHeap> m_samplerHeap;  
ComPtr<ID3D12PipelineState> m_pipelineState1;  
ComPtr<ID3D12PipelineState> m_pipelineState2;  
D3D12_RECT m_scissorRect;
```

Checking supported features.

C++

```
inline UINT8 D3D12GetFormatPlaneCount(  
    _In_ ID3D12Device* pDevice,  
    DXGI_FORMAT Format  
)  
{  
    D3D12_FEATURE_DATA_FORMAT_INFO formatInfo = {Format};  
    if (FAILED(pDevice->CheckFeatureSupport(D3D12_FEATURE_FORMAT_INFO,  
&formatInfo, sizeof(formatInfo))))  
    {  
        return 0;  
    }  
    return formatInfo.PlaneCount;  
}
```

Refer to the Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptors](#)

[ID3D12Device1](#)

[ID3D12Device2](#)

[ID3D12Object](#)

[Memory Management in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CheckFeatureSupport method (d3d12.h)

Article 02/22/2024

Gets information about the features that are supported by the current graphics driver.

Syntax

C++

```
HRESULT CheckFeatureSupport(
    D3D12_FEATURE Feature,
    [in, out] void* pFeatureSupportData,
    UINT FeatureSupportDataSize
);
```

Parameters

Feature

Type: [D3D12_FEATURE](#)

A constant from the [D3D12_FEATURE](#) enumeration describing the feature(s) that you want to query for support.

[in, out] pFeatureSupportData

Type: [void*](#)

A pointer to a data structure that corresponds to the value of the *Feature* parameter. To determine the corresponding data structure for each constant, see [D3D12_FEATURE](#).

FeatureSupportDataSize

Type: [UINT](#)

The size of the structure pointed to by the *pFeatureSupportData* parameter.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful. Returns **E_INVALIDARG** if an unsupported data type is passed to the *pFeatureSupportData* parameter or if a size mismatch is detected for the *FeatureSupportDataSize* parameter.

Remarks

As a usage example, to check for ray tracing support, specify the [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) structure in the *pFeatureSupportData* parameter. When the function completes successfully, access the *RaytracingTier* field (which specifies the supported ray tracing tier) of the now-populated [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) structure.

For more info, see [Capability Querying](#).

Hardware support for DXGI Formats

To view tables of DXGI formats and hardware features, refer to:

- [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](#)
- [Format Support for Direct3D Feature Level 10.1 Hardware](#)
- [Format Support for Direct3D Feature Level 10.0 Hardware](#)

Examples

The [D3D1211on12](#) sample uses [ID3D12Device::CheckFeatureSupport](#) as follows:

C++

```
inline UINT8 D3D12GetFormatPlaneCount(
    _In_ ID3D12Device* pDevice,
    DXGI_FORMAT Format
)
{
    D3D12_FEATURE_DATA_FORMAT_INFO formatInfo = {Format};
    if (FAILED(pDevice->CheckFeatureSupport(D3D12_FEATURE_FORMAT_INFO,
&formatInfo, sizeof(formatInfo))))
    {
        return 0;
    }
    return formatInfo.PlaneCount;
```

}

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CopyDescriptors method (d3d12.h)

Article 02/22/2024

Copies descriptors from a source to a destination.

Syntax

C++

```
void CopyDescriptors(
    [in]          UINT           NumDestDescriptorRanges,
    [in]          const D3D12_CPU_DESCRIPTOR_HANDLE*
    *pDestDescriptorRangeStarts,
    [in, optional] const UINT
    *pDestDescriptorRangeSizes,
    [in]          UINT           NumSrcDescriptorRanges,
    [in]          const D3D12_CPU_DESCRIPTOR_HANDLE*
    *pSrcDescriptorRangeStarts,
    [in, optional] const UINT
    *pSrcDescriptorRangeSizes,
    [in]          D3D12_DESCRIPTOR_HEAP_TYPE      DescriptorHeapsType
);
```

Parameters

[in] NumDestDescriptorRanges

Type: [UINT](#)

The number of destination descriptor ranges to copy to.

[in] pDestDescriptorRangeStarts

Type: [const D3D12_CPU_DESCRIPTOR_HANDLE*](#)

An array of [D3D12_CPU_DESCRIPTOR_HANDLE](#) objects to copy to.

All the destination and source descriptors must be in heaps of the same [D3D12_DESCRIPTOR_HEAP_TYPE](#).

[in, optional] pDestDescriptorRangeSizes

Type: [const UINT*](#)

An array of destination descriptor range sizes to copy to.

[in] NumSrcDescriptorRanges

Type: **UINT**

The number of source descriptor ranges to copy from.

[in] pSrcDescriptorRangeStarts

Type: **const D3D12_CPU_DESCRIPTOR_HANDLE***

An array of **D3D12_CPU_DESCRIPTOR_HANDLE** objects to copy from.

ⓘ Important

All elements in the *pSrcDescriptorRangeStarts* parameter must be in a non shader-visible descriptor heap. This is because shader-visible descriptor heaps may be created in **WRITE_COMBINE** memory or GPU local memory, which is prohibitively slow to read from. If your application manages descriptor heaps via copying the descriptors required for a given pass or frame from local "storage" descriptor heaps to the GPU-bound descriptor heap, use shader-opaque heaps for the storage heaps and copy into the GPU-visible heap as required.

[in, optional] pSrcDescriptorRangeSizes

Type: **const UINT***

An array of source descriptor range sizes to copy from.

[in] DescriptorHeapsType

Type: **D3D12_DESCRIPTOR_HEAP_TYPE**

The **D3D12_DESCRIPTOR_HEAP_TYPE**-typed value that specifies the type of descriptor heap to copy with. This is required as different descriptor types may have different sizes.

Both the source and destination descriptor heaps must have the same type, else the debug layer will emit an error.

Return value

None

Remarks

Where applicable, prefer [ID3D12Device::CopyDescriptorsSimple](#) to this method. It can have a better CPU cache miss rate due to the linear nature of the copy.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Copying Descriptors](#)

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CopyDescriptorsSimple method (d3d12.h)

Article 02/22/2024

Copies descriptors from a source to a destination.

Syntax

C++

```
void CopyDescriptorsSimple(
    [in] UINT NumDescriptors,
    [in] D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptorRangeStart,
    [in] D3D12_CPU_DESCRIPTOR_HANDLE SrcDescriptorRangeStart,
    [in] D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapsType
);
```

Parameters

[in] NumDescriptors

Type: [UINT](#)

The number of descriptors to copy.

[in] DestDescriptorRangeStart

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A [D3D12_CPU_DESCRIPTOR_HANDLE](#) that describes the destination descriptors to start to copy to.

The destination and source descriptors must be in heaps of the same [D3D12_DESCRIPTOR_HEAP_TYPE](#).

[in] SrcDescriptorRangeStart

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A [D3D12_CPU_DESCRIPTOR_HANDLE](#) that describes the source descriptors to start to copy from.

ⓘ Important

The `SrcDescriptorRangeStart` parameter must be in a non shader-visible descriptor heap. This is because shader-visible descriptor heaps may be created in `WRITE_COMBINE` memory or GPU local memory, which is prohibitively slow to read from. If your application manages descriptor heaps via copying the descriptors required for a given pass or frame from local "storage" descriptor heaps to the GPU-bound descriptor heap, then use shader-opaque heaps for the storage heaps and copy into the GPU-visible heap as required.

[in] `DescriptorHeapsType`

Type: [D3D12_DESCRIPTOR_HEAP_TYPE](#)

The `D3D12_DESCRIPTOR_HEAP_TYPE`-typed value that specifies the type of descriptor heap to copy with. This is required as different descriptor types may have different sizes.

Both the source and destination descriptor heaps must have the same type, else the debug layer will emit an error.

Return value

None

Remarks

Where applicable, prefer this method to [ID3D12Device::CopyDescriptors](#). It can have a better CPU cache miss rate due to the linear nature of the copy.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Copying Descriptors](#)

[ID3D12Device](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateCommandAllocator method (d3d12.h)

Article02/22/2024

Creates a command allocator object.

Syntax

C++

```
HRESULT CreateCommandAllocator(
    [in] D3D12_COMMAND_LIST_TYPE type,
    REFIID                 riid,
    [out] void              **ppCommandAllocator
);
```

Parameters

[in] type

Type: [D3D12_COMMAND_LIST_TYPE](#)

A [D3D12_COMMAND_LIST_TYPE](#)-typed value that specifies the type of command allocator to create. The type of command allocator can be the type that records either direct command lists or bundles.

riid

Type: [REFIID](#)

The globally unique identifier ([GUID](#)) for the command allocator interface ([ID3D12CommandAllocator](#)). The [REFIID](#), or [GUID](#), of the interface to the command allocator can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12CommandAllocator)` will get the [GUID](#) of the interface to a command allocator.

[out] ppCommandAllocator

Type: [void**](#)

A pointer to a memory block that receives a pointer to the [ID3D12CommandAllocator](#) interface for the command allocator.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the command allocator. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

The device creates command lists from the command allocator.

Examples

The [D3D12Bundles](#) sample uses [ID3D12Device::CreateCommandAllocator](#) as follows:

C++

```
ThrowIfFailed(pDevice->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,
    IID_PPV_ARGS(&m_commandAllocator)));
ThrowIfFailed(pDevice->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_BUNDLE,
    IID_PPV_ARGS(&m_bundleAllocator)));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[\[\]](#) [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateCommandList method (d3d12.h)

Article 10/13/2021

Creates a command list.

Syntax

C++

```
HRESULT CreateCommandList(
    [in]          UINT           nodeMask,
    [in]          D3D12_COMMAND_LIST_TYPE type,
    [in]          ID3D12CommandAllocator *pCommandAllocator,
    [in, optional] ID3D12PipelineState   *pInitialState,
    [in]          REFIID         riid,
    [out]         void           **ppCommandList
);
```

Parameters

[in] nodeMask

Type: [UINT](#)

For single-GPU operation, set this to zero. If there are multiple GPU nodes, then set a bit to identify the node (the device's physical adapter) for which to create the command list. Each bit in the mask corresponds to a single node. Only one bit must be set. Also see [Multi-adapter systems](#).

[in] type

Type: [D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of command list to create.

[in] pCommandAllocator

Type: [ID3D12CommandAllocator*](#)

A pointer to the command allocator object from which the device creates command lists.

[in, optional] `pInitialState`

Type: [ID3D12PipelineState*](#)

An optional pointer to the pipeline state object that contains the initial pipeline state for the command list. If it is `nullptr`, then the runtime sets a dummy initial pipeline state, so that drivers don't have to deal with undefined state. The overhead for this is low, particularly for a command list, for which the overall cost of recording the command list likely dwarfs the cost of a single initial state setting. So there's little cost in not setting the initial pipeline state parameter, if doing so is inconvenient.

For bundles, on the other hand, it might make more sense to try to set the initial state parameter (since bundles are likely smaller overall, and can be reused frequently).

[in] `riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the command list interface to return in `ppCommandList`.

[out] `ppCommandList`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the [ID3D12CommandList](#) or [ID3D12GraphicsCommandList](#) interface for the command list.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

 Expand table

Return value	Description
<code>E_OUTOFMEMORY</code>	There is insufficient memory to create the command list.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

The device creates command lists from the command allocator.

Examples

The [D3D12Bundles](#) sample uses `ID3D12Device::CreateCommandList` as follows.

Create the pipeline objects.

C++

```
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
```

Create a command allocator.

C++

```
ThrowIfFailed(m_device-
    >CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,
    IID_PPV_ARGS(&m_commandAllocator)));
```

Creating the direct command list.

C++

```
ThrowIfFailed(m_device->CreateCommandList(0, D3D12_COMMAND_LIST_TYPE_DIRECT,
m_commandAllocator.Get(), nullptr, IID_PPV_ARGS(&m_commandList)));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[ID3D12GraphicsCommandList::Reset](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateCommandQueue method (d3d12.h)

Article 02/22/2024

Creates a command queue.

Also see [ID3D12Device9::CreateCommandQueue1](#).

Syntax

C++

```
HRESULT CreateCommandQueue(
    const D3D12_COMMAND_QUEUE_DESC *pDesc,
    REFIID                      riid,
    void                         **ppCommandQueue
);
```

Parameters

pDesc

Type: [in] [const D3D12_COMMAND_QUEUE_DESC*](#)

Specifies a [D3D12_COMMAND_QUEUE_DESC](#) that describes the command queue.

riid

Type: [REFIID](#)

The globally unique identifier (GUID) for the command queue interface. See [Remarks](#).

An input parameter.

ppCommandQueue

Type: [out] [void**](#)

A pointer to a memory block that receives a pointer to the [ID3D12CommandQueue](#) interface for the command queue.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the command queue. See [Direct3D 12 return codes](#) for other possible return values.

Remarks

The **REFIID**, or **GUID**, of the interface to the command queue can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12CommandQueue)` will get the **GUID** of the interface to a command queue.

Examples

The [D3D12HelloTriangle](#) sample uses **ID3D12Device::CreateCommandQueue** as follows:

C++

```
D3D12_COMMAND_QUEUE_DESC queueDesc{};  
queueDesc.Flags = D3D12_COMMAND_QUEUE_FLAG_NONE;  
queueDesc.Type = D3D12_COMMAND_LIST_TYPE_DIRECT;  
  
ThrowIfFailed(m_device->CreateCommandQueue(&queueDesc,  
IID_PPV_ARGS(&m_commandQueue)));
```

Refer to the [Example code in the D3D12 reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12Device](#)

- ID3D12Device9::CreateCommandQueue1
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateCommandSignature method (d3d12.h)

Article 02/22/2024

This method creates a command signature.

Syntax

C++

```
HRESULT CreateCommandSignature(
    [in]          const D3D12_COMMAND_SIGNATURE_DESC *pDesc,
    [in, optional] ID3D12RootSignature           *pRootSignature,
                           REFIID                   riid,
    [out, optional] void*                         **ppvCommandSignature
);
```

Parameters

[in] pDesc

Type: [const D3D12_COMMAND_SIGNATURE_DESC*](#)

Describes the command signature to be created with the [D3D12_COMMAND_SIGNATURE_DESC](#) structure.

[in, optional] pRootSignature

Type: [ID3D12RootSignature*](#)

Specifies the [ID3D12RootSignature](#) that the command signature applies to.

The root signature is required if any of the commands in the signature will update bindings on the pipeline. If the only command present is a draw or dispatch, the root signature parameter can be set to NULL.

riid

Type: [REFIID](#)

The globally unique identifier ([GUID](#)) for the command signature interface ([ID3D12CommandSignature](#)). The [REFIID](#), or [GUID](#), of the interface to the command

signature can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12CommandSignature)` will get the **GUID** of the interface to a command signature.

[out, optional] `ppvCommandSignature`

Type: **void****

Specifies a pointer, that on successful completion of the method will point to the created command signature ([ID3D12CommandSignature](#)).

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateCommittedResource method (d3d12.h)

Article 10/13/2021

Creates both a resource and an implicit heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.

Syntax

C++

```
HRESULT CreateCommittedResource(
    [in]          const D3D12_HEAP_PROPERTIES *pHeapProperties,
    [in]          D3D12_HEAP_FLAGS           HeapFlags,
    [in]          const D3D12_RESOURCE_DESC *pDesc,
    [in]          D3D12_RESOURCE_STATES   InitialResourceState,
    [in, optional] const D3D12_CLEAR_VALUE *pOptimizedClearValue,
    [in]          REFIID                 riidResource,
    [out, optional] void                  **ppvResource
);
```

Parameters

[in] pHeapProperties

Type: [const D3D12_HEAP_PROPERTIES*](#)

A pointer to a **D3D12_HEAP_PROPERTIES** structure that provides properties for the resource's heap.

[in] HeapFlags

Type: [D3D12_HEAP_FLAGS](#)

Heap options, as a bitwise-OR'd combination of **D3D12_HEAP_FLAGS** enumeration constants.

[in] pDesc

Type: [const D3D12_RESOURCE_DESC*](#)

A pointer to a **D3D12_RESOURCE_DESC** structure that describes the resource.

[in] InitialResourceState

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

When you create a resource together with a [D3D12_HEAP_TYPE_UPLOAD](#) heap, you must set *InitialResourceState* to [D3D12_RESOURCE_STATE_GENERIC_READ](#).

When you create a resource together with a [D3D12_HEAP_TYPE_READBACK](#) heap, you must set *InitialResourceState* to [D3D12_RESOURCE_STATE_COPY_DEST](#).

[in, optional] pOptimizedClearValue

Type: [const D3D12_CLEAR_VALUE*](#)

Specifies a [D3D12_CLEAR_VALUE](#) structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use [D3D12_RESOURCE_DIMENSION_BUFFER](#), you must set *pOptimizedClearValue* to `nullptr`.

[in] riidResource

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the resource interface to return in *ppvResource*.

While *riidResource* is most commonly the GUID of [ID3D12Resource](#), it may be the GUID of any interface. If the resource object doesn't support the interface for this GUID, then creation fails with [E_NOINTERFACE](#).

[out, optional] ppvResource

Type: [void**](#)

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and `S_FALSE` is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
<code>E_OUTOFMEMORY</code>	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

This method creates both a resource and a heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. The created heap is known as an implicit heap, because the heap object can't be obtained by the application. Before releasing the final reference on the resource, your application must ensure that the GPU will no longer read nor write to this resource.

The implicit heap is made resident for GPU access before the method returns control to your application. Also see [Residency](#).

The resource GPU VA mapping can't be changed. See [ID3D12CommandQueue::UpdateTileMappings](#) and [Volume tiled resources](#).

This method may be called by multiple threads concurrently.

Examples

The [D3D12Bundles](#) sample uses `ID3D12Device::CreateCommittedResource` as follows:

Create a vertex buffer.

C++

```
auto heapProperties = CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_DEFAULT);
auto resourceDesc =
CD3DX12_RESOURCE_DESC::Buffer(SampleAssets::VertexDataSize);
ThrowIfFailed(m_device->CreateCommittedResource(
    &heapProperties,
    D3D12_HEAP_FLAG_NONE,
    &resourceDesc,
    D3D12_RESOURCE_STATE_COPY_DEST,
    nullptr,
    IID_PPV_ARGS(&m_vertexBuffer)));
```

See Example code in the Direct3D 12 reference.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[CreatePlacedResource](#)

[CreateReservedResource](#)

[D3D12_HEAP_FLAGS](#)

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateComputePipelineState method (d3d12.h)

Article 02/22/2024

Creates a compute pipeline state object.

Syntax

C++

```
HRESULT CreateComputePipelineState(
    [in]  const D3D12_COMPUTE_PIPELINE_STATE_DESC *pDesc,
          REFIID                           riid,
    [out] void*                         **ppPipelineState
);
```

Parameters

[in] pDesc

Type: **const D3D12_COMPUTE_PIPELINE_STATE_DESC***

A pointer to a [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structure that describes compute pipeline state.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the pipeline state interface ([ID3D12PipelineState](#)). The **REFIID**, or **GUID**, of the interface to the pipeline state can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12PipelineState)` will get the **GUID** of the interface to a pipeline state.

[out] ppPipelineState

Type: **void****

A pointer to a memory block that receives a pointer to the [ID3D12PipelineState](#) interface for the pipeline state object. The pipeline state object is an immutable state object. It contains no methods.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the pipeline state object. See [Direct3D 12 Return Codes](#) for other possible return values.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateConstantBufferView method (d3d12.h)

Article 02/22/2024

Creates a constant-buffer view for accessing resource data.

Syntax

C++

```
void CreateConstantBufferView(
    [in, optional] const D3D12_CONSTANT_BUFFER_VIEW_DESC *pDesc,
    [in]           D3D12_CPU_DESCRIPTOR_HANDLE           DestDescriptor
);
```

Parameters

[in, optional] pDesc

Type: [const D3D12_CONSTANT_BUFFER_VIEW_DESC*](#)

A pointer to a [D3D12_CONSTANT_BUFFER_VIEW_DESC](#) structure that describes the constant-buffer view.

[in] DestDescriptor

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap that holds the constant-buffer view.

Return value

None

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateDepthStencilView method (d3d12.h)

Article 02/22/2024

Creates a depth-stencil view for accessing resource data.

Syntax

C++

```
void CreateDepthStencilView(
    [in, optional] ID3D12Resource                  *pResource,
    [in, optional] const D3D12_DEPTH_STENCIL_VIEW_DESC *pDesc,
    [in]           D3D12_CPU_DESCRIPTOR_HANDLE        DestDescriptor
);
```

Parameters

[in, optional] pResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) object that represents the depth stencil.

At least one of *pResource* or *pDesc* must be provided. A null *pResource* is used to initialize a null descriptor, which guarantees D3D11-like null binding behavior (reading 0s, writes are discarded), but must have a valid *pDesc* in order to determine the descriptor type.

[in, optional] pDesc

Type: [const D3D12_DEPTH_STENCIL_VIEW_DESC*](#)

A pointer to a [D3D12_DEPTH_STENCIL_VIEW_DESC](#) structure that describes the depth-stencil view.

A null *pDesc* is used to initialize a default descriptor, if possible. This behavior is identical to the D3D11 null descriptor behavior, where defaults are filled in. This behavior inherits the resource format and dimension (if not typeless) and DSVs target the first mip and all array slices. Not all resources support null descriptor initialization.

[in] DestDescriptor

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap that holds the depth-stencil view.

Return value

None

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateDescriptorHeap method (d3d12.h)

Article 02/22/2024

Creates a descriptor heap object.

Syntax

C++

```
HRESULT CreateDescriptorHeap(
    [in]  const D3D12_DESCRIPTOR_HEAP_DESC *pDescriptorHeapDesc,
          REFIID                      riid,
    [out] void*                  **ppvHeap
);
```

Parameters

[in] pDescriptorHeapDesc

Type: **const D3D12_DESCRIPTOR_HEAP_DESC***

A pointer to a **D3D12_DESCRIPTOR_HEAP_DESC** structure that describes the heap.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the descriptor heap interface. See Remarks. An input parameter.

[out] ppvHeap

Type: **void****

A pointer to a memory block that receives a pointer to the descriptor heap. *ppvHeap* can be NULL, to enable capability testing. When *ppvHeap* is NULL, no object will be created and S_FALSE will be returned when *pDescriptorHeapDesc* is valid.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the descriptor heap object. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

The **REFIID**, or **GUID**, of the interface to the descriptor heap can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12DescriptorHeap)` will get the **GUID** of the interface to a descriptor heap.

Examples

The [D3D12HelloWorld](#) sample uses `ID3D12Device::CreateDescriptorHeap` as follows:

Describe and create a render target view (RTV) descriptor heap.

C++

```
// Create descriptor heaps.
{
    // Describe and create a render target view (RTV) descriptor heap.
    D3D12_DESCRIPTOR_HEAP_DESC rtvHeapDesc = {};
    rtvHeapDesc.NumDescriptors = FrameCount;
    rtvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_RTV;
    rtvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
    ThrowIfFailed(m_device->CreateDescriptorHeap(&rtvHeapDesc,
        IID_PPV_ARGS(&m_rtvHeap)));

    m_rtvDescriptorSize = m_device-
        >GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_RTV);
}

// Create frame resources.
{
    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
        >GetCPUDescriptorHandleForHeapStart());

    // Create a RTV for each frame.
    for (UINT n = 0; n < FrameCount; n++)
    {
        ThrowIfFailed(m_swapChain->GetBuffer(n,
            IID_PPV_ARGS(&m_renderTargets[n])));
        m_device->CreateRenderTargetView(m_renderTargets[n].Get(), nullptr,
            rtvHandle);
        rtvHandle.Offset(1, m_rtvDescriptorSize);
    }
}
```

Refer to the Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateFence method (d3d12.h)

Article 02/22/2024

Creates a fence object.

Syntax

C++

```
HRESULT CreateFence(
    UINT64           InitialValue,
    D3D12_FENCE_FLAGS Flags,
    REFIID            riid,
    [out] void**      ppFence
);
```

Parameters

InitialValue

Type: [UINT64](#)

The initial value for the fence.

Flags

Type: [D3D12_FENCE_FLAGS](#)

A combination of [D3D12_FENCE_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies options for the fence.

riid

Type: [REFIID](#)

The globally unique identifier (**GUID**) for the fence interface ([ID3D12Fence](#)). The **REFIID**, or **GUID**, of the interface to the fence can be obtained by using the `_uuidof()` macro.

For example, `_uuidof(ID3D12Fence)` will get the **GUID** of the interface to a fence.

[out] ppFence

Type: **void****

A pointer to a memory block that receives a pointer to the [ID3D12Fence](#) interface that is used to access the fence.

Return value

Type: [HRESULT](#)

Returns [S_OK](#) if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12Device::CreateGraphicsPipelineState method (d3d12.h)

Article 02/22/2024

Creates a graphics pipeline state object.

Syntax

C++

```
HRESULT CreateGraphicsPipelineState(
    [in] const D3D12_GRAPHICS_PIPELINE_STATE_DESC *pDesc,
    REFIID riid,
    [out] void **ppPipelineState
);
```

Parameters

[in] pDesc

Type: **const D3D12_GRAPHICS_PIPELINE_STATE_DESC***

A pointer to a **D3D12_GRAPHICS_PIPELINE_STATE_DESC** structure that describes graphics pipeline state.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the pipeline state interface (**ID3D12PipelineState**). The **REFIID**, or **GUID**, of the interface to the pipeline state can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12PipelineState)` will get the **GUID** of the interface to a pipeline state.

[out] ppPipelineState

Type: **void****

A pointer to a memory block that receives a pointer to the **ID3D12PipelineState** interface for the pipeline state object. The pipeline state object is an immutable state object. It contains no methods.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the pipeline state object. See [Direct3D 12 Return Codes](#) for other possible return values.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateHeap method (d3d12.h)

Article 10/13/2021

Creates a heap that can be used with placed resources and reserved resources.

Syntax

C++

```
HRESULT CreateHeap(  
    [in]          const D3D12_HEAP_DESC *pDesc,  
    [in]          REFIID             riid,  
    [out, optional] void             **ppvHeap  
) ;
```

Parameters

[in] pDesc

Type: **const D3D12_HEAP_DESC***

A pointer to a constant **D3D12_HEAP_DESC** structure that describes the heap.

[in] riid

Type: **REFIID**

A reference to the globally unique identifier (**GUID**) of the heap interface to return in *ppvHeap*.

While *riidResource* is most commonly the **GUID** of **ID3D12Heap**, it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with **E_NOINTERFACE**.

[out, optional] ppvHeap

Type: **void****

An optional pointer to a memory block that receives the requested interface pointer to the created heap object.

ppvHeap can be `nullptr`, to enable capability testing. When *ppvHeap* is `nullptr`, no object is created, and `S_FALSE` is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
<code>E_OUTOFMEMORY</code>	There is insufficient memory to create the heap.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

`CreateHeap` creates a heap that can be used with placed resources and reserved resources.

Before releasing the final reference on the heap, your application must ensure that the GPU will no longer read or write to this heap.

A placed resource object holds a reference on the heap it is created on; but a reserved resource doesn't hold a reference for each mapping made to a heap.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	<code>d3d12.h</code>
Library	<code>D3D12.lib</code>
DLL	<code>D3D12.dll</code>

See also

[ID3D12Device](#)

[Shared heaps](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreatePlacedResource method (d3d12.h)

Article02/10/2023

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

Your application can re-use video memory by overlapping multiple Direct3D placed and reserved resources on heap regions. The simple memory re-use model (described in [Remarks](#)) exists to clarify which overlapping resource is valid at any given time. To maximize graphics tool support, with the simple model data-inheritance isn't supported; and finer-grained tile and sub-resource invalidation isn't supported. Only full overlapping resource invalidation occurs.

Syntax

C++

```
HRESULT CreatePlacedResource(
    ID3D12Heap                 *pHeap,
    UINT64                      HeapOffset,
    const D3D12_RESOURCE_DESC   *pDesc,
    D3D12_RESOURCE_STATES      InitialState,
    const D3D12_CLEAR_VALUE     *pOptimizedClearValue,
    REFIID                      riid,
    void                        **ppvResource
);
```

Parameters

pHeap

Type: [in] [ID3D12Heap*](#)

A pointer to the [ID3D12Heap](#) interface that represents the heap in which the resource is placed.

HeapOffset

Type: [UINT64](#)

The offset, in bytes, to the resource. The *HeapOffset* must be a multiple of the resource's alignment, and *HeapOffset* plus the resource size must be smaller than or equal to the heap size. [GetResourceAllocationInfo](#) must be used to understand the sizes of texture resources.

`pDesc`

Type: [in] const [D3D12_RESOURCE_DESC](#)*

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

`InitialState`

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

When a resource is created together with a [D3D12_HEAP_TYPE_UPLOAD](#) heap, *InitialState* must be [D3D12_RESOURCE_STATE_GENERIC_READ](#). When a resource is created together with a [D3D12_HEAP_TYPE_READBACK](#) heap, *InitialState* must be [D3D12_RESOURCE_STATE_COPY_DEST](#).

`pOptimizedClearValue`

Type: [in, optional] const [D3D12_CLEAR_VALUE](#)*

Specifies a [D3D12_CLEAR_VALUE](#) that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, your application should choose the value that the clear operation will most commonly be called with.

Clear operations can be called with other values, but those operations will not be as efficient as when the value matches the one passed into resource creation.

pOptimizedClearValue must be NULL when used with [D3D12_RESOURCE_DIMENSION_BUFFER](#).

`riid`

Type: [REFIID](#)

The globally unique identifier (**GUID**) for the resource interface. This is an input parameter.

The **REFIID**, or **GUID**, of the interface to the resource can be obtained by using the `_uuidof` macro. For example, `_uuidof(ID3D12Resource)` gets the **GUID** of the interface to a resource. Although **riid** is, most commonly, the **GUID** for **ID3D12Resource**, it may be any **GUID** for any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with **E_NOINTERFACE**.

`ppvResource`

Type: [out, optional] **void****

A pointer to a memory block that receives a pointer to the resource. *ppvResource* can be **NULL**, to enable capability testing. When *ppvResource* is **NULL**, no object will be created and **S_FALSE** will be returned when *pResourceDesc* and other parameters are valid.

Return value

Type: **HRESULT**

This method returns **E_OUTOFMEMORY** if there is insufficient memory to create the resource. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

CreatePlacedResource is similar to fully mapping a reserved resource to an offset within a heap; but the virtual address space associated with a heap may be reused as well.

Placed resources are lighter weight to create and destroy than committed resources are. This is because no heap is created nor destroyed during those operations. In addition, placed resources enable an even lighter weight technique to reuse memory than resource creation and destruction—that is, reuse through aliasing, and aliasing barriers. Multiple placed resources may simultaneously overlap each other on the same heap, but only a single overlapping resource can be used at a time.

There are two placed resource usage semantics—a simple model, and an advanced model. We recommend that you choose the simple model (it maximizes graphics tool support across the diverse ecosystem of GPUs), unless and until you find that you need the advanced model for your app.

Simple model

In this model, you can consider a placed resource to be in one of two states: active, or inactive. It's invalid for the GPU to either read or write from an inactive resource. Placed resources are created in the inactive state.

To activate a resource with an aliasing barrier on a command list, your application must pass the resource in [D3D12_RESOURCE_ALIASING_BARRIER::pResourceAfter](#).

pResourceBefore can be left NULL during an activation. All resources that share physical memory with the activated resource now become inactive, which includes overlapping placed and reserved resources.

Aliasing barriers should be grouped up and submitted together, in order to maximize efficiency.

After activation, resources with either the render target or depth stencil flags must be further initialized. See the notes on the required resource initialization below.

Notes on the required resource initialization

Certain resource types still require initialization. Resources with either the render target or depth stencil flags must be initialized with either a clear operation or a collection of full subresource copies. If an aliasing barrier was used to denote the transition between two aliased resources, the initialization must occur after the aliasing barrier. This initialization is still required whenever a resource would've been activated in the simple model.

Placed and reserved resources with either the render target or depth stencil flags must be initialized with one of the following operations before other operations are supported.

- A *Clear* operation; for example [ClearRenderTargetView](#) or [ClearDepthStencilView](#).
- A [DiscardResource](#) operation.
- A *Copy* operation; for example [CopyBufferRegion](#), [CopyTextureRegion](#), or [CopyResource](#).

Applications should prefer the most explicit operation that results in the least amount of texels modified. Consider the following examples.

- Using a depth buffer to solve pixel visibility typically requires each depth texel start out at 1.0 or 0. Therefore, a *Clear* operation should be the most efficient option for aliased depth buffer initialization.
- An application may use an aliased render target as a destination for tone mapping. Since the application will render over every pixel during the tone mapping, [DiscardResource](#) should be the most efficient option for initialization.

Advanced model

In this model, you can ignore the active/inactive state abstraction. Instead, you must honor these lower-level rules.

- An aliasing barrier must be between two different GPU resource accesses of the same physical memory, as long as those accesses are within the same [ExecuteCommandLists](#) call.
- The first rendering operation to certain types of aliased resource must still be an initialization, just like the simple model.

Initialization operations must occur either on an entire subresource, or on a 64KB granularity. An entire subresource initialization is supported for all resource types. A 64KB initialization granularity, aligned at a 64KB offset, is supported for buffers and textures with either the 64KB_UNDEFINED_SWIZZLE or 64KB_STANDARD_SWIZZLE texture layout (refer to [D3D12_TEXTURE_LAYOUT](#)).

Notes on the aliasing barrier

The aliasing barrier may set NULL for both *pResourceAfter* and *pResourceBefore*. The memory coherence definition of [ExecuteCommandLists](#) and an aliasing barrier are the same, such that two aliased accesses to the same physical memory need no aliasing barrier when the accesses are in two different [ExecuteCommandLists](#) invocations.

For D3D12 advanced usage models, the synchronization definition of [ExecuteCommandLists](#) is equivalent to an aliasing barrier. Therefore, applications may either insert an aliasing barrier between reusing physical memory, or ensure the two aliased usages of physical memory occurs in two separate calls to [ExecuteCommandLists](#).

The amount of inactivation varies based on resource properties. Textures with undefined memory layouts are the worst case, as the entire texture must be inactivated atomically. For two overlapping resources with defined layouts, inactivation can result in only the overlapping aligned regions of a resource. Data inheritance can even be well-defined. For more details, see [Memory aliasing and data inheritance](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[CreateCommittedResource](#)

[CreateReservedResource](#)

[ID3D12Device](#)

[Shared Heaps](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateQueryHeap method (d3d12.h)

Article 02/22/2024

Creates a query heap. A query heap contains an array of queries.

Syntax

C++

```
HRESULT CreateQueryHeap(  
    [in]           const D3D12_QUERY_HEAP_DESC *pDesc,  
                           REFIID                iid,  
    [out, optional] void        **ppvHeap  
) ;
```

Parameters

[in] pDesc

Type: **const D3D12_QUERY_HEAP_DESC***

Specifies the query heap in a **D3D12_QUERY_HEAP_DESC** structure.

iid

Type: **REFIID**

Specifies a REFIID that uniquely identifies the heap.

[out, optional] ppvHeap

Type: **void****

Specifies a pointer to the heap, that will be returned on successful completion of the method. *ppvHeap* can be NULL, to enable capability testing. When *ppvHeap* is NULL, no object will be created and S_FALSE will be returned when *pDesc* is valid.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Refer to [Queries](#) for more information.

Examples

The [D3D12PredicationQueries](#) sample uses `ID3D12Device::CreateQueryHeap` as follows:

Create a query heap and a query result buffer.

C++

```
// Pipeline objects.  
D3D12_VIEWPORT m_viewport;  
D3D12_RECT m_scissorRect;  
ComPtr<IDXGISwapChain3> m_swapChain;  
ComPtr<ID3D12Device> m_device;  
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];  
ComPtr<ID3D12CommandAllocator> m_commandAllocators[FrameCount];  
ComPtr<ID3D12CommandQueue> m_commandQueue;  
ComPtr<ID3D12RootSignature> m_rootSignature;  
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;  
ComPtr<ID3D12DescriptorHeap> m_cbvHeap;  
ComPtr<ID3D12DescriptorHeap> m_dsvHeap;  
ComPtr<ID3D12QueryHeap> m_queryHeap;  
UINT m_rtvDescriptorSize;  
UINT m_cbvSrvDescriptorSize;  
UINT m_frameIndex;  
  
// Synchronization objects.  
ComPtr<ID3D12Fence> m_fence;  
UINT64 m_fenceValues[FrameCount];  
HANDLE m_fenceEvent;  
  
// Asset objects.  
ComPtr<ID3D12PipelineState> m_pipelineState;  
ComPtr<ID3D12PipelineState> m_queryState;  
ComPtr<ID3D12GraphicsCommandList> m_commandList;  
ComPtr<ID3D12Resource> m_vertexBuffer;  
ComPtr<ID3D12Resource> m_constantBuffer;  
ComPtr<ID3D12Resource> m_depthStencil;  
ComPtr<ID3D12Resource> m_queryResult;  
D3D12_VERTEX_BUFFER_VIEW m_vertexBufferView;
```

C++

```
// Describe and create a heap for occlusion queries.  
D3D12_QUERY_HEAP_DESC queryHeapDesc = {};  
queryHeapDesc.Count = 1;  
queryHeapDesc.Type = D3D12_QUERY_HEAP_TYPE_OCCLUSION;  
ThrowIfFailed(m_device->CreateQueryHeap(&queryHeapDesc,  
IID_PPV_ARGS(&m_queryHeap)));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateRenderTargetView method (d3d12.h)

Article 02/22/2024

Creates a render-target view for accessing resource data.

Syntax

C++

```
void CreateRenderTargetView(
    [in, optional] ID3D12Resource                 *pResource,
    [in, optional] const D3D12_RENDER_TARGET_VIEW_DESC *pDesc,
    [in]           D3D12_CPU_DESCRIPTOR_HANDLE        DestDescriptor
);
```

Parameters

[in, optional] pResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) object that represents the render target.

At least one of *pResource* or *pDesc* must be provided. A null *pResource* is used to initialize a null descriptor, which guarantees D3D11-like null binding behavior (reading 0s, writes are discarded), but must have a valid *pDesc* in order to determine the descriptor type.

[in, optional] pDesc

Type: [const D3D12_RENDER_TARGET_VIEW_DESC*](#)

A pointer to a [D3D12_RENDER_TARGET_VIEW_DESC](#) structure that describes the render-target view.

A null *pDesc* is used to initialize a default descriptor, if possible. This behavior is identical to the D3D11 null descriptor behavior, where defaults are filled in. This behavior inherits the resource format and dimension (if not typeless) and RTVs target the first mip and all array slices. Not all resources support null descriptor initialization.

[in] DestDescriptor

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the destination where the newly-created render target view will reside.

Return value

None

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateReservedResource method (d3d12.h)

Article 10/13/2021

Creates a resource that is reserved, and not yet mapped to any pages in a heap.

Syntax

C++

```
HRESULT CreateReservedResource(
    [in]           const D3D12_RESOURCE_DESC *pDesc,
    [in]           D3D12_RESOURCE_STATES   InitialState,
    [in, optional] const D3D12_CLEAR_VALUE *pOptimizedClearValue,
    [in]           REFIID                riid,
    [out, optional] void                 **ppvResource
);
```

Parameters

[in] pDesc

Type: [const D3D12_RESOURCE_DESC*](#)

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

[in] InitialState

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

[in, optional] pOptimizedClearValue

Type: [const D3D12_CLEAR_VALUE*](#)

Specifies a [D3D12_CLEAR_VALUE](#) structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the

`D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET` or `D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL` flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use `D3D12_RESOURCE_DIMENSION_BUFFER`, you must set `pOptimizedClearValue` to `nullptr`.

`[in] riid`

Type: `REFIID`

A reference to the globally unique identifier (**GUID**) of the resource interface to return in `ppvResource`. See **Remarks**.

While `riidResource` is most commonly the **GUID** of `ID3D12Resource`, it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with `E_NOINTERFACE`.

`[out, optional] ppvResource`

Type: `void**`

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

`ppvResource` can be `nullptr`, to enable capability testing. When `ppvResource` is `nullptr`, no object is created, and `S_FALSE` is returned when `pDesc` is valid.

Return value

Type: `HRESULT`

If the function succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT error code](#).

[] Expand table

Return value	Description
<code>E_OUTOFMEMORY</code>	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

`CreateReservedResource` is equivalent to [D3D11_RESOURCE_MISC_TILED](#) in Direct3D 11. It creates a resource with virtual memory only, no backing store.

You need to map the resource to physical memory (that is, to a heap) using [CopyTileMappings](#) and [UpdateTileMappings](#).

These resource types can only be created when the adapter supports tiled resource tier 1 or greater. The tiled resource tier defines the behavior of accessing a resource that is not mapped to a heap.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[CreateCommittedResource](#)

[CreatePlacedResource](#)

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12Device::CreateRootSignature method (d3d12.h)

Article 10/13/2021

Creates a root signature layout.

Syntax

C++

```
HRESULT CreateRootSignature(
    [in]    UINT      nodeMask,
    [in]    const void *pBlobWithRootSignature,
    [in]    SIZE_T    blobLengthInBytes,
    [in]    REFIID    riid,
    [out]   void     **ppvRootSignature
);
```

Parameters

[in] nodeMask

Type: [UINT](#)

For single GPU operation, set this to zero. If there are multiple GPU nodes, set bits to identify the nodes (the device's physical adapters) to which the root signature is to apply. Each bit in the mask corresponds to a single node. Refer to [Multi-adapter systems](#).

[in] pBlobWithRootSignature

Type: [const void*](#)

A pointer to the source data for the serialized signature.

[in] blobLengthInBytes

Type: [SIZE_T](#)

The size, in bytes, of the block of memory that *pBlobWithRootSignature* points to.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the root signature interface. See Remarks. An input parameter.

[out] `ppvRootSignature`

Type: **void****

A pointer to a memory block that receives a pointer to the root signature.

Return value

Type: **HRESULT**

Returns **S_OK** if successful; otherwise, returns one of the [Direct3D 12 Return Codes](#).

This method returns **E_INVALIDARG** if the blob that *pBlobWithRootSignature* points to is invalid.

Remarks

If an application procedurally generates a [D3D12_ROOT_SIGNATURE_DESC](#) data structure, it must pass a pointer to this **D3D12_ROOT_SIGNATURE_DESC** in a call to [D3D12SerializeRootSignature](#) to make the serialized form. The application then passes the serialized form to *pBlobWithRootSignature* in a call to [ID3D12Device::CreateRootSignature](#).

The **REFIID**, or **GUID**, of the interface to the root signature layout can be obtained by using the `__uuidof()` macro. For example, `__uuidof(ID3D12RootSignature)` will get the **GUID** of the interface to a root signature.

Examples

The [D3D12HelloTriangle](#) sample uses [ID3D12Device::CreateRootSignature](#) as follows:

Create an empty root signature.

C++

```
CD3DX12_ROOT_SIGNATURE_DESC rootSignatureDesc;
rootSignatureDesc.Init(0, nullptr, 0, nullptr,
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);
```

```
ComPtr<ID3DBlob> signature;
ComPtr<ID3DBlob> error;
ThrowIfFailed(D3D12SerializeRootSignature(&rootSignatureDesc,
D3D_ROOT_SIGNATURE_VERSION_1, &signature, &error));
ThrowIfFailed(m_device->CreateRootSignature(0, signature-
>GetBufferPointer(), signature->GetBufferSize(),
IID_PPV_ARGS(&m_rootSignature)));
```

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateSampler method (d3d12.h)

Article 02/22/2024

Create a sampler object that encapsulates sampling information for a texture.

Syntax

C++

```
void CreateSampler(  
    [in] const D3D12_SAMPLER_DESC     *pDesc,  
    [in] D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor  
);
```

Parameters

[in] pDesc

Type: [const D3D12_SAMPLER_DESC*](#)

A pointer to a [D3D12_SAMPLER_DESC](#) structure that describes the sampler.

[in] DestDescriptor

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap that holds the sampler.

Return value

None

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateShaderResourceView method (d3d12.h)

Article 07/27/2022

Creates a shader-resource view for accessing data in a resource.

Syntax

C++

```
void CreateShaderResourceView(
    [in, optional] ID3D12Resource                  *pResource,
    [in, optional] const D3D12_SHADER_RESOURCE_VIEW_DESC *pDesc,
    [in]           D3D12_CPU_DESCRIPTOR_HANDLE        DestDescriptor
);
```

Parameters

[in, optional] pResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) object that represents the shader resource.

At least one of *pResource* or *pDesc* must be provided. A null *pResource* is used to initialize a null descriptor, which guarantees D3D11-like null binding behavior (reading 0s, writes are discarded), but must have a valid *pDesc* in order to determine the descriptor type.

[in, optional] pDesc

Type: [const D3D12_SHADER_RESOURCE_VIEW_DESC*](#)

A pointer to a [D3D12_SHADER_RESOURCE_VIEW_DESC](#) structure that describes the shader-resource view.

A null *pDesc* is used to initialize a default descriptor, if possible. This behavior is identical to the D3D11 null descriptor behavior, where defaults are filled in. This behavior inherits the resource format and dimension (if not typeless) and for buffers SRVs target a full buffer and are typed (not raw or structured), and for textures SRVs target a full texture, all mips and all array slices. Not all resources support null descriptor initialization.

[in] DestDescriptor

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the shader-resource view. This handle can be created in a shader-visible or non-shader-visible descriptor heap.

Return value

None

Remarks

Processing YUV 4:2:0 video formats

An app must map the luma (Y) plane separately from the chroma (UV) planes. Developers do this by calling [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.

YUV 4:2:0 formats are listed in [DXGI_FORMAT](#).

Examples

The [D3D12nBodyGravity](#) sample uses [ID3D12Device::CreateShaderResourceView](#) as follows:

Describe and create two shader resource views based on one description.

C++

```
D3D12_SHADER_RESOURCE_VIEW_DESC srvDesc = {};
srvDesc.Shader4ComponentMapping = D3D12_DEFAULT_SHADER_4_COMPONENT_MAPPING;
srvDesc.Format = DXGI_FORMAT_UNKNOWN;
srvDesc.ViewDimension = D3D12_SRV_DIMENSION_BUFFER;
srvDesc.Buffer.FirstElement = 0;
srvDesc.Buffer.NumElements = ParticleCount;
srvDesc.Buffer.StructureByteStride = sizeof(Particle);
srvDesc.Buffer.Flags = D3D12_BUFFER_SRV_FLAG_NONE;

CD3DX12_CPU_DESCRIPTOR_HANDLE srvHandle0(m\_svrUavHeap->GetCPUDescriptorHandleForHeapStart\(\), SrvParticlePosVelo0 + index,
```

```
m_srvUavDescriptorSize);  
CD3DX12_CPU_DESCRIPTOR_HANDLE srvHandle1(m_srvUavHeap-  
>GetCPUDescriptorHandleForHeapStart(), SrvParticlePosVelo1 + index,  
m_srvUavDescriptorSize);  
m_device->CreateShaderResourceView(m_particleBuffer0[index].Get(), &srvDesc,  
srvHandle0);  
m_device->CreateShaderResourceView(m_particleBuffer1[index].Get(), &srvDesc,  
srvHandle1);
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateSharedHandle method (d3d12.h)

Article 10/13/2021

Creates a shared handle to a heap, resource, or fence object.

Syntax

C++

```
HRESULT CreateSharedHandle(
    [in]          ID3D12DeviceChild      *pObject,
    [in, optional] const SECURITY_ATTRIBUTES *pAttributes,
                           DWORD             Access,
    [in, optional] LPCWSTR            Name,
    [out]           HANDLE            *pHandle
);
```

Parameters

[in] pObject

Type: [ID3D12DeviceChild*](#)

A pointer to the [ID3D12DeviceChild](#) interface that represents the heap, resource, or fence object to create for sharing. The following interfaces (derived from [ID3D12DeviceChild](#)) are supported:

- [ID3D12Heap](#)
- [ID3D12Resource](#)
- [ID3D12Fence](#)

[in, optional] pAttributes

Type: [const SECURITY_ATTRIBUTES*](#)

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 **IpSecurityDescriptor** 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](#).

Access

Type: [DWORD](#)

Currently the only value this parameter accepts is [GENERIC_ALL](#).

[in, optional] Name

Type: [LPCWSTR](#)

A **NULL**-terminated **UNICODE** string that contains the name to associate with the shared heap. The name is limited to [MAX_PATH](#) characters. Name comparison is case-sensitive.

If *Name* 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

Type: [HANDLE*](#)

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

Type: [HRESULT](#)

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 [Direct3D 12 Return Codes](#) topic.

Remarks

Both heaps and committed resources can be shared. Sharing a committed resource shares the implicit heap along with the committed resource description, such that a compatible resource description can be mapped to the heap from another device.

For Direct3D 11 and Direct3D 12 interop scenarios, a shared fence is opened in DirectX 11 with the [ID3D11Device5::OpenSharedFence](#) method, and a shared resource is opened with the [ID3D11Device::OpenSharedResource1](#) method.

For Direct3D 12, a shared handle is opened with the [ID3D12Device::OpenSharedHandle](#) or the [ID3D12Device::OpenSharedHandleByName](#) method.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::CreateUnorderedAccessView method (d3d12.h)

Article 10/13/2021

Creates a view for unordered accessing.

Syntax

C++

```
void CreateUnorderedAccessView(
    ID3D12Resource                  *pResource,
    ID3D12Resource                  *pCounterResource,
    const D3D12_UNORDERED_ACCESS_VIEW_DESC *pDesc,
    [in] D3D12_CPU_DESCRIPTOR_HANDLE   DestDescriptor
);
```

Parameters

pResource

Type: [in, optional] [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) object that represents the unordered access.

At least one of *pResource* or *pDesc* must be provided.

A null *pResource* is used to initialize a null descriptor, which guarantees Direct3D 11-like null binding behavior (reading 0s, writes are discarded), but must have a valid *pDesc* in order to determine the descriptor type.

pCounterResource

Type: [in, optional] [ID3D12Resource*](#)

The [ID3D12Resource](#) for the counter (if any) associated with the UAV.

If *pCounterResource* is not specified, then the [CounterOffsetInBytes](#) member of the [D3D12_BUFFER_UAV](#) structure must be 0.

If *pCounterResource* is specified, then there is a counter associated with the UAV, and the runtime performs validation of the following requirements:

- The **StructureByteStride** member of the [D3D12_BUFFER_UAV](#) structure must be greater than 0.
- The format must be [DXGI_FORMAT_UNKNOWN](#).
- The [D3D12_BUFFER_UAV_FLAG_RAW](#) flag (a [D3D12_BUFFER_UAV_FLAGS](#) enumeration constant) must not be set.
- Both of the resources (*pResource* and *pCounterResource*) must be buffers.
- The **CounterOffsetInBytes** member of the [D3D12_BUFFER_UAV](#) structure must be a multiple of [**D3D12_UAV_COUNTER_PLACEMENT_ALIGNMENT**](#) (4096), and must be within the range of the counter resource.
- *pResource* cannot be NULL
- *pDesc* cannot be NULL.

`pDesc`

Type: [in, optional] const [D3D12_UNORDERED_ACCESS_VIEW_DESC](#)*

A pointer to a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure that describes the unordered-access view.

A null *pDesc* is used to initialize a default descriptor, if possible. This behavior is identical to the D3D11 null descriptor behavior, where defaults are filled in. This behavior inherits the resource format and dimension (if not typeless) and for buffers UAVs target a full buffer and are typed, and for textures UAVs target the first mip and all array slices. Not all resources support null descriptor initialization.

[in] `DestDescriptor`

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap that holds the unordered-access view.

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::Evict method (d3d12.h)

Article 10/13/2021

Enables the page-out of data, which precludes GPU access of that data.

Syntax

C++

```
HRESULT Evict(  
    UINT           NumObjects,  
    [in] ID3D12Pageable * const *ppObjects  
);
```

Parameters

NumObjects

Type: [UINT](#)

The number of objects in the *ppObjects* array to evict from the device.

[in] ppObjects

Type: [ID3D12Pageable*](#)

A pointer to a memory block that contains an array of [ID3D12Pageable](#) interface pointers for the objects.

Even though most D3D12 objects inherit from [ID3D12Pageable](#), residency changes are only supported on the following objects: Descriptor Heaps, Heaps, Committed Resources, and Query Heaps

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Evict persists the data associated with a resource to disk, and then removes the resource from the memory pool where it was located. This method should be called on the object which owns the physical memory: either a committed resource (which owns both virtual and physical memory assignments) or a heap - noting that reserved resources do not have physical memory, and placed resources are borrowing memory from a heap.

Refer to the remarks for [MakeResident](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetAdapterLuid method (d3d12.h)

Article 02/22/2024

Gets a locally unique identifier for the current device (adapter).

Syntax

C++

```
LUID GetAdapterLuid();
```

Return value

Type: [LUID](#)

The locally unique identifier for the adapter.

Remarks

This method returns a unique identifier for the adapter that is specific to the adapter hardware. Applications can use this identifier to define robust mappings across various APIs (Direct3D 12, DXGI).

A locally unique identifier (LUID) is a 64-bit value that is guaranteed to be unique only on the system on which it was generated. The uniqueness of a locally unique identifier (LUID) is guaranteed only until the system is restarted.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

Requirement	Value
DLL	D3d12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetCopyableFootprints method (d3d12.h)

Article 10/13/2021

Gets a resource layout that can be copied. Helps the app fill-in [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#) and [D3D12_SUBRESOURCE_FOOTPRINT](#) when suballocating space in upload heaps.

Syntax

C++

```
void GetCopyableFootprints(
    [in]          const D3D12_RESOURCE_DESC* pResourceDesc,
    [in]          UINT FirstSubresource,
    [in]          UINT NumSubresources,
    [in]          UINT64 BaseOffset,
    [out, optional] D3D12_PLACED_SUBRESOURCE_FOOTPRINT* pLayouts,
    [out, optional] UINT* pNumRows,
    [out, optional] UINT64* pRowSizeInBytes,
    [out, optional] UINT64* pTotalBytes
);
```

Parameters

[in] pResourceDesc

Type: [const D3D12_RESOURCE_DESC*](#)

A description of the resource, as a pointer to a [D3D12_RESOURCE_DESC](#) structure.

[in] FirstSubresource

Type: [UINT](#)

Index of the first subresource in the resource. The range of valid values is 0 to [D3D12_REQ_SUBRESOURCES](#).

[in] NumSubresources

Type: [UINT](#)

The number of subresources in the resource. The range of valid values is 0 to (`D3D12_REQ_SUBRESOURCES` - *FirstSubresource*).

`BaseOffset`

Type: `UINT64`

The offset, in bytes, to the resource.

`[out, optional] pLayouts`

Type: `D3D12_PLACED_SUBRESOURCE_FOOTPRINT*`

A pointer to an array (of length *NumSubresources*) of `D3D12_PLACED_SUBRESOURCE_FOOTPRINT` structures, to be filled with the description and placement of each subresource.

`[out, optional] pNumRows`

Type: `UINT*`

A pointer to an array (of length *NumSubresources*) of integer variables, to be filled with the number of rows for each subresource.

`[out, optional] pRowSizeInBytes`

Type: `UINT64*`

A pointer to an array (of length *NumSubresources*) of integer variables, each entry to be filled with the unpadded size in bytes of a row, of each subresource.

For example, if a Texture2D resource has a width of 32 and bytes per pixel of 4,

then *pRowSizeInBytes* returns 128.

pRowSizeInBytes should not be confused with **row pitch**, as examining *pLayouts* and getting the row pitch from that will give you 256 as it is aligned to `D3D12_TEXTURE_DATA_PITCH_ALIGNMENT`.

`[out, optional] pTotalBytes`

Type: `UINT64*`

A pointer to an integer variable, to be filled with the total size, in bytes.

Return value

None

Remarks

This routine assists the application in filling out [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#) and [D3D12_SUBRESOURCE_FOOTPRINT](#) structures, when suballocating space in upload heaps. The resulting structures are GPU adapter-agnostic, meaning that the values will not vary from one GPU adapter to the next. [GetCopyableFootprints](#) uses specified details about resource formats, texture layouts, and alignment requirements (from the [D3D12_RESOURCE_DESC](#) structure) to fill out the subresource structures. Applications have access to all these details, so this method, or a variation of it, could be written as part of the app.

Examples

The [D3D12Multithreading](#) sample uses [ID3D12Device::GetCopyableFootprints](#) as follows:

C++

```
// Returns required size of a buffer to be used for data upload
inline UINT64 GetRequiredIntermediateSize(
    _In_ ID3D12Resource* pDestinationResource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES) UINT FirstSubresource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES-FirstSubresource) UINT
NumSubresources)
{
    D3D12_RESOURCE_DESC Desc = pDestinationResource->GetDesc();
    UINT64 RequiredSize = 0;

    ID3D12Device* pDevice;
    pDestinationResource->GetDevice(__uuidof(*pDevice),
    reinterpret_cast<void**>(&pDevice));
    pDevice->GetCopyableFootprints(&Desc, FirstSubresource, NumSubresources,
0, nullptr, nullptr, nullptr, &RequiredSize);
    pDevice->Release();

    return RequiredSize;
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CD3DX12_RESOURCE_DESC](#)

[CD3DX12_SUBRESOURCE_FOOTPRINT](#)

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetCustomHeapProperties(UINT,D3D12_HEAP_TYPE) method (d3d12.h)

Article 06/12/2024

Divulges the equivalent custom heap properties that are used for non-custom heap types, based on the adapter's architectural properties.

Syntax

C++

```
D3D12_HEAP_PROPERTIES GetCustomHeapProperties(  
    [in] UINT             nodeMask,  
    D3D12_HEAP_TYPE      heapType  
)
```

Parameters

[in] nodeMask

Type: **UINT**

For single-GPU operation, set this to zero. If there are multiple GPU nodes, set a bit to identify the node (the device's physical adapter). Each bit in the mask corresponds to a single node. Only 1 bit must be set. See [Multi-adapter systems](#).

heapType

Type: **D3D12_HEAP_TYPE**

A **D3D12_HEAP_TYPE**-typed value that specifies the heap to get properties for. **D3D12_HEAP_TYPE_CUSTOM** is not supported as a parameter value.

Return value

Type: **D3D12_HEAP_PROPERTIES**

Returns a **D3D12_HEAP_PROPERTIES** structure that provides properties for the specified heap. The **Type** member of the returned **D3D12_HEAP_PROPERTIES** is always

D3D12_HEAP_TYPE_CUSTOM.

When D3D12_FEATURE_DATA_ARCHITECTURE::UMA is FALSE, the returned D3D12_HEAP_PROPERTIES members convert as follows:

[+] Expand table

Heap Type	How the returned D3D12_HEAP_PROPERTIES members convert
D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L1.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.

When D3D12_FEATURE_DATA_ARCHITECTURE::UMA is TRUE and D3D12_FEATURE_DATA_ARCHITECTURE::CacheCoherentUMA is FALSE, the returned D3D12_HEAP_PROPERTIES members convert as follows:

[+] Expand table

Heap Type	How the returned D3D12_HEAP_PROPERTIES members convert
D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_COMBINE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.

When D3D12_FEATURE_DATA_ARCHITECTURE::UMA is TRUE and D3D12_FEATURE_DATA_ARCHITECTURE::CacheCoherentUMA is TRUE, the returned D3D12_HEAP_PROPERTIES members convert as follows:

[+] Expand table

Heap Type	How the returned D3D12_HEAP_PROPERTIES members

convert

D3D12_HEAP_TYPE_UPLOAD	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_DEFAULT	CPUPageProperty = NOT_AVAILABLE, MemoryPoolPreference = L0.
D3D12_HEAP_TYPE_READBACK	CPUPageProperty = WRITE_BACK, MemoryPoolPreference = L0.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetDescriptorHandleIncrementSize method (d3d12.h)

Article 02/22/2024

Gets the size of the handle increment for the given type of descriptor heap. This value is typically used to increment a handle into a descriptor array by the correct amount.

Syntax

C++

```
UINT GetDescriptorHandleIncrementSize(  
    [in] D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapType  
) ;
```

Parameters

[in] DescriptorHeapType

The [D3D12_DESCRIPTOR_HEAP_TYPE](#)-typed value that specifies the type of descriptor heap to get the size of the handle increment for.

Return value

Returns the size of the handle increment for the given type of descriptor heap, including any necessary padding.

Remarks

The descriptor size returned by this method is used as one input to the helper structures [CD3DX12_CPU_DESCRIPTOR_HANDLE](#) and [CD3DX12_GPU_DESCRIPTOR_HANDLE](#).

Examples

The [D3D12PredicationQueries](#) sample uses `ID3D12Device::GetDescriptorHandleIncrementSize` as follows:

Create the descriptor heap for the resources. The `m_rtvDescriptorSize` variable stores the render target view descriptor handle increment size, and is used in the **Create frame resources** section of the code.

C++

```
// Create descriptor heaps.
{
    // Describe and create a render target view (RTV) descriptor heap.
    D3D12_DESCRIPTOR_HEAP_DESC rtvHeapDesc = {};
    rtvHeapDesc.NumDescriptors = FrameCount;
    rtvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_RTV;
    rtvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
    ThrowIfFailed(m_device->CreateDescriptorHeap(&rtvHeapDesc,
        IID_PPV_ARGS(&m_rtvHeap)));

    // Describe and create a depth stencil view (DSV) descriptor heap.
    D3D12_DESCRIPTOR_HEAP_DESC dsvHeapDesc = {};
    dsvHeapDesc.NumDescriptors = 1;
    dsvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_DSV;
    dsvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_NONE;
    ThrowIfFailed(m_device->CreateDescriptorHeap(&dsvHeapDesc,
        IID_PPV_ARGS(&m_dsvHeap)));

    // Describe and create a constant buffer view (CBV) descriptor heap.
    D3D12_DESCRIPTOR_HEAP_DESC cbvHeapDesc = {};
    cbvHeapDesc.NumDescriptors = CbvCountPerFrame * FrameCount;
    cbvHeapDesc.Type = D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV;
    cbvHeapDesc.Flags = D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE;
    ThrowIfFailed(m_device->CreateDescriptorHeap(&cbvHeapDesc,
        IID_PPV_ARGS(&m_cbvHeap)));

    // Describe and create a heap for occlusion queries.
    D3D12_QUERY_HEAP_DESC queryHeapDesc = {};
    queryHeapDesc.Count = 1;
    queryHeapDesc.Type = D3D12_QUERY_HEAP_TYPE_OCCLUSION;
    ThrowIfFailed(m_device->CreateQueryHeap(&queryHeapDesc,
        IID_PPV_ARGS(&m_queryHeap)));

    m_rtvDescriptorSize = m_device-
        >GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_RTV);
    m_cbvSrvDescriptorSize = m_device-
        >GetDescriptorHandleIncrementSize(D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV);
}

// Create frame resources.
{
    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
        >GetCPUDescriptorHandleForHeapStart());

    // Create a RTV and a command allocator for each frame.
    for (UINT n = 0; n < FrameCount; n++)
    {
```

```

        ThrowIfFailed(m_swapChain->GetBuffer(n,
IID_PPV_ARGS(&m_renderTargets[n])));
    m_device->CreateRenderTargetView(m_renderTargets[n].Get(), nullptr,
rtvHandle);
    rtvHandle.Offset(1, m_rtvDescriptorSize);

    ThrowIfFailed(m_device-
>CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,
IID_PPV_ARGS(&m_commandAllocators[n])));
}
}

```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetDeviceRemovedReason method (d3d12.h)

Article 02/22/2024

Gets the reason that the device was removed, or **S_OK** if the device isn't removed. To be called back when a device is removed, consider using [ID3D12Fence::SetEventOnCompletion](#) with a value of **UINT64_MAX**. That's because device removal causes all fences to be signaled to that value (which also implies completing all events waited on, because they'll all be less than **UINT64_MAX**).

Syntax

C++

```
HRESULT GetDeviceRemovedReason();
```

Return value

Type: [HRESULT](#)

This method returns the reason that the device was removed.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetNodeCount method (d3d12.h)

Article 02/22/2024

Reports the number of physical adapters (nodes) that are associated with this device.

Syntax

C++

```
UINT GetNodeCount();
```

Return value

Type: [UINT](#)

The number of physical adapters (nodes) that this device has.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12Device::GetResourceAllocationInfo(UINT,UINT,const D3D12_RESOURCE_DESC*) method (d3d12.h)

Article 06/12/2024

Gets the size and alignment of memory required for a collection of resources on this adapter.

Syntax

C++

```
D3D12_RESOURCE_ALLOCATION_INFO GetResourceAllocationInfo(  
    [in] UINT                     visibleMask,  
    [in] UINT                     numResourceDescs,  
    [in] const D3D12_RESOURCE_DESC *pResourceDescs  
) ;
```

Parameters

[in] `visibleMask`

Type: [UINT](#)

For single-GPU operation, set this to zero. If there are multiple GPU nodes, then set bits to identify the nodes (the device's physical adapters). Each bit in the mask corresponds to a single node. Also see [Multi-adapter systems](#).

[in] `numResourceDescs`

Type: [UINT](#)

The number of resource descriptors in the `pResourceDescs` array.

[in] `pResourceDescs`

Type: [const D3D12_RESOURCE_DESC*](#)

An array of `D3D12_RESOURCE_DESC` structures that described the resources to get info about.

Return value

Type: [D3D12_RESOURCE_ALLOCATION_INFO](#)

A [D3D12_RESOURCE_ALLOCATION_INFO](#) structure that provides info about video memory allocated for the specified array of resources.

If an error occurs, then [D3D12_RESOURCE_ALLOCATION_INFO::SizeInBytes](#) equals [UINT64_MAX](#).

Remarks

When you're using [CreatePlacedResource](#), your application must use [GetResourceAllocationInfo](#) in order to understand the size and alignment characteristics of texture resources. The results of this method vary depending on the particular adapter, and must be treated as unique to this adapter and driver version.

Your application can't use the output of [GetResourceAllocationInfo](#) to understand packed mip properties of textures. To understand packed mip properties of textures, your application must use [GetResourceTiling](#).

Texture resource sizes significantly differ from the information returned by [GetResourceTiling](#), because some adapter architectures allocate extra memory for textures to reduce the effective bandwidth during common rendering scenarios. This even includes textures that have constraints on their texture layouts, or have standardized texture layouts. That extra memory can't be sparsely mapped nor remapped by an application using [CreateReservedResource](#) and [UpdateTileMappings](#), so it isn't reported by [GetResourceTiling](#).

Your application can forgo using [GetResourceAllocationInfo](#) for buffer resources ([D3D12_RESOURCE_DIMENSION_BUFFER](#)). Buffers have the same size on all adapters, which is merely the smallest multiple of 64KB that's greater or equal to [D3D12_RESOURCE_DESC::Width](#).

When multiple resource descriptions are passed in, the C++ algorithm for calculating a structure size and alignment are used. For example, a three-element array with two tiny 64KB-aligned resources and a tiny 4MB-aligned resource, reports differing sizes based on the order of the array. If the 4MB aligned resource is in the middle, then the resulting **Size** is 12MB. Otherwise, the resulting **Size** is 8MB. The **Alignment** returned would always be 4MB, because it's the superset of all alignments in the resource array.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::GetResourceTiling method (d3d12.h)

Article 07/27/2022

Gets info about how a tiled resource is broken into tiles.

Syntax

C++

```
void GetResourceTiling(
    [in]                  ID3D12Resource          *pTiledResource,
    [out, optional]        UINT                 *pNumTilesForEntireResource,
    [out, optional]        D3D12_PACKED_MIP_INFO *pPackedMipDesc,
    [out, optional]        D3D12_TILE_SHAPE
    *pStandardTileShapeForNonPackedMips,
    [in, out, optional]   UINT                 *pNumSubresourceTilings,
    [in]                  UINT                 FirstSubresourceTilingToGet,
    [out]                 D3D12_SUBRESOURCE_TILING
    *pSubresourceTilingsForNonPackedMips
);
```

Parameters

[in] pTiledResource

Type: [ID3D12Resource*](#)

Specifies a tiled [ID3D12Resource](#) to get info about.

[out, optional] pNumTilesForEntireResource

Type: [UINT*](#)

A pointer to a variable that receives the number of tiles needed to store the entire tiled resource.

[out, optional] pPackedMipDesc

Type: [D3D12_PACKED_MIP_INFO*](#)

A pointer to a [D3D12_PACKED_MIP_INFO](#) structure that [GetResourceTiling](#) fills with info about how the tiled resource's mipmaps are packed.

[out, optional] *pStandardTileShapeForNonPackedMips*

Type: [D3D12_TILE_SHAPE*](#)

Specifies a [D3D12_TILE_SHAPE](#) structure that [GetResourceTiling](#) fills with info about the tile shape. This is info about how pixels fit in the tiles, independent of tiled resource's dimensions, not including packed mipmaps. If the entire tiled resource is packed, this parameter is meaningless because the tiled resource has no defined layout for packed mipmaps. In this situation, [GetResourceTiling](#) sets the members of [D3D12_TILE_SHAPE](#) to zeros.

[in, out, optional] *pNumSubresourceTilings*

Type: [UINT*](#)

A pointer to a variable that contains the number of tiles in the subresource. On input, this is the number of subresources to query tilings for; on output, this is the number that was actually retrieved at *pSubresourceTilingsForNonPackedMips* (clamped to what's available).

[in] *FirstSubresourceTilingToGet*

Type: [UINT](#)

The number of the first subresource tile to get. [GetResourceTiling](#) ignores this parameter if the number that *pNumSubresourceTilings* points to is 0.

[out] *pSubresourceTilingsForNonPackedMips*

Type: [D3D12_SUBRESOURCE_TILING*](#)

Specifies a [D3D12_SUBRESOURCE_TILING](#) structure that [GetResourceTiling](#) fills with info about subresource tiles. If subresource tiles are part of packed mipmaps, [GetResourceTiling](#) sets the members of [D3D12_SUBRESOURCE_TILING](#) to zeros, except the *StartTileIndexInOverallResource* member, which [GetResourceTiling](#) sets to [D3D12_PACKED_TILE](#) (0xffffffff). The [D3D12_PACKED_TILE](#) constant indicates that the whole [D3D12_SUBRESOURCE_TILING](#) structure is meaningless for this situation, and the info that the *pPackedMipDesc* parameter points to applies.

Return value

None

Remarks

To estimate the total resource size of textures needed when calculating heap sizes and calling [CreatePlacedResource](#), use [GetResourceAllocationInfo](#) instead of [GetResourceTiling](#). [GetResourceTiling](#) cannot be used for this.

For more information on tiled resources, refer to [Volume Tiled Resources](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device](#)

[Subresources](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::MakeResident method (d3d12.h)

Article 10/13/2021

Makes objects resident for the device.

Syntax

C++

```
HRESULT MakeResident(
    UINT           NumObjects,
    [in] ID3D12Pageable * const *ppObjects
);
```

Parameters

NumObjects

Type: **UINT**

The number of objects in the *ppObjects* array to make resident for the device.

[in] ppObjects

Type: **ID3D12Pageable***

A pointer to a memory block that contains an array of **ID3D12Pageable** interface pointers for the objects.

Even though most D3D12 objects inherit from **ID3D12Pageable**, residency changes are only supported on the following objects: Descriptor Heaps, Heaps, Committed Resources, and Query Heaps

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

MakeResident loads the data associated with a resource from disk, and re-allocates the memory from the resource's appropriate memory pool. This method should be called on the object which owns the physical memory.

Use this method, and [Evict](#), to manage GPU video memory, noting that this was done automatically in D3D11, but now has to be done by the app in D3D12.

MakeResident and [Evict](#) can help applications manage the residency budget on many adapters. **MakeResident** explicitly pages-in data and, then, precludes page-out so the GPU can access the data. [Evict](#) enables page-out.

Some GPU architectures do not benefit from residency manipulation, due to the lack of sufficient GPU virtual address space. Use

[D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT](#) and

[IDXGIAdapter3::QueryVideoMemoryInfo](#) to recognize when the maximum GPU VA space per-process is too small or roughly the same size as the residency budget. For such architectures, the residency budget will always be constrained by the amount of GPU virtual address space. [Evict](#) will not free-up any residency budget on such systems.

Applications must handle **MakeResident** failures, even if there appears to be enough residency budget available. Physical memory fragmentation and adapter architecture quirks can preclude the utilization of large contiguous ranges. Applications should free up more residency budget before trying again.

MakeResident is ref-counted, such that [Evict](#) must be called the same amount of times as **MakeResident** before [Evict](#) takes effect. Objects that support residency are made resident during creation, so a single [Evict](#) call will actually evict the object.

Applications must use fences to ensure the GPU doesn't use non-resident objects.

MakeResident must return before the GPU executes a command list that references the object. [Evict](#) must be called after the GPU finishes executing a command list that references the object.

Evicted objects still consume the same GPU virtual address and same amount of GPU virtual address space. Therefore, resource descriptors and other GPU virtual address references are not invalidated after [Evict](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::OpenSharedHandle method (d3d12.h)

Article 02/22/2024

Opens a handle for shared resources, shared heaps, and shared fences, by using HANDLE and REFIID.

Syntax

C++

```
HRESULT OpenSharedHandle(  
    [in]          HANDLE NTHandle,  
                REFIID riid,  
    [out, optional] void   **ppvObj  
) ;
```

Parameters

[in] NTHandle

Type: **HANDLE**

The handle that was output by the call to [ID3D12Device::CreateSharedHandle](#).

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for one of the following interfaces:

- [ID3D12Heap](#)
- [ID3D12Resource](#)
- [ID3D12Fence](#)

The **REFIID**, or **GUID**, of the interface can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12Heap)` will get the **GUID** of the interface to a resource.

[out, optional] ppvObj

Type: **void****

A pointer to a memory block that receives a pointer to one of the following interfaces:

- [ID3D12Heap](#)
- [ID3D12Resource](#)
- [ID3D12Fence](#)

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[Multi-adapter systems](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::OpenSharedHandleByName method (d3d12.h)

Article02/22/2024

Opens a handle for shared resources, shared heaps, and shared fences, by using Name and Access.

Syntax

C++

```
HRESULT OpenSharedHandleByName(
    [in]  LPCWSTR Name,
          DWORD   Access,
    [out] HANDLE  *pNTHandle
);
```

Parameters

[in] Name

Type: **LPCWSTR**

The name that was optionally passed as the *Name* parameter in the call to [ID3D12Device::CreateSharedHandle](#).

Access

Type: **DWORD**

The access level that was specified in the *Access* parameter in the call to [ID3D12Device::CreateSharedHandle](#).

[out] pNTHandle

Type: **HANDLE***

Pointer to the shared handle.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device::SetStablePowerState method (d3d12.h)

Article02/22/2024

A development-time aid for certain types of profiling and experimental prototyping.

Syntax

C++

```
HRESULT SetStablePowerState(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: **BOOL**

Specifies a **BOOL** that turns the stable power state on or off.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method is only useful during the development of applications. It enables developers to profile GPU usage of multiple algorithms without experiencing artifacts from [dynamic frequency scaling](#).

Do not call this method in normal execution for a shipped application. This method only works while the machine is in [developer mode](#). If developer mode is not enabled, then device removal will occur. Instead, call this method in response to an off-by-default, developer-facing switch. Calling it in response to command line parameters, config files, registry keys, and developer console commands are reasonable usage scenarios.

A stable power state typically fixes GPU clock rates at a slower setting that is significantly lower than that experienced by users under normal application load. This reduction in clock rate affects the entire system. Slow clock rates are required to ensure processors don't exhaust power, current, and thermal limits. Normal usage scenarios commonly leverage a processor's ability to dynamically over-clock. Any conclusions made by comparing two designs under a stable power state should be double-checked with supporting results from real usage scenarios.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device1 interface (d3d12.h)

Article02/22/2024

Represents a virtual adapter, and expands on the range of methods provided by [ID3D12Device](#).

Note This interface was introduced in Windows 10 Anniversary Update.

Applications targeting Windows 10 Anniversary Update should use this interface instead of earlier or later versions. Applications targeting an earlier or later version of Windows 10 should use the appropriate version of the [ID3D12Device](#) interface.

Inheritance

The [ID3D12Device1](#) interface inherits from [ID3D12Device](#). [ID3D12Device1](#) also has these types of members:

Methods

The [ID3D12Device1](#) interface has these methods.

[+] [Expand table](#)

[ID3D12Device1::CreatePipelineLibrary](#)

Creates a cached pipeline library.

[ID3D12Device1::SetEventOnMultipleFenceCompletion](#)

Specifies an event that should be fired when one or more of a collection of fences reach specific values.

[ID3D12Device1::SetResidencyPriority](#)

This method sets residency priorities of a specified list of objects.

Remarks

Use [D3D12CreateDevice](#) to create a device.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

[ID3D12Device2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device1::CreatePipelineLibrary method (d3d12.h)

Article 08/03/2021

Creates a cached pipeline library. For pipeline state objects (PSOs) that are expected to share data together, grouping them into a library before serializing them means that there's less overhead due to metadata, as well as the opportunity to avoid redundant or duplicated data being written to disk.

You can query for **ID3D12PipelineLibrary** support with **ID3D12Device::CheckFeatureSupport**, with **D3D12_FEATURE_SHADER_CACHE** and **D3D12_FEATURE_DATA_SHADER_CACHE**. If the *Flags* member of **D3D12_FEATURE_DATA_SHADER_CACHE** contains the flag **D3D12_SHADER_CACHE_SUPPORT_LIBRARY**, the **ID3D12PipelineLibrary** interface is supported. If not, then **DXGI_ERROR_NOT_SUPPORTED** will always be returned when this function is called.

Syntax

C++

```
HRESULT CreatePipelineLibrary(
    const void *pLibraryBlob,
    SIZE_T     BlobLength,
    REFIID     riid,
    void       **ppPipelineLibrary
);
```

Parameters

`pLibraryBlob`

Type: [in] **const void***

If the input library blob is empty, then the initial content of the library is empty. If the input library blob is not empty, then it is validated for integrity, parsed, and the pointer is stored. The pointer provided as input to this method must remain valid for the lifetime of the object returned. For efficiency reasons, the data is not copied.

`BlobLength`

Type: [SIZE_T](#)

Specifies the length of *pLibraryBlob* in bytes.

`riid`

Type: [REFIID](#)

Specifies a unique REFIID for the [ID3D12PipelineLibrary](#) object. Typically set this and the following parameter with the macro `IID_PPV_ARGS(&Library)`, where **Library** is the name of the object.

`ppPipelineLibrary`

Type: [out] [void**](#)

Returns a pointer to the created library.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns **S_OK**. Otherwise, it returns an [HRESULT error code](#), including **E_INVALIDARG** if the blob is corrupted or unrecognized, **D3D12_ERROR_DRIVER_VERSION_MISMATCH** if the provided data came from an old driver or runtime, and **D3D12_ERROR_ADAPTER_NOT_FOUND** if the data came from different hardware.

If you pass `nullptr` for *pPipelineLibrary* then the runtime still performs the validation of the blob but avoid creating the actual library and returns **S_FALSE** if the library would have been created.

Also, the feature requires an updated driver, and attempting to use it on old drivers will return **DXGI_ERROR_UNSUPPORTED**.

Remarks

A pipeline library enables the following operations.

- Adding pipeline state objects (PSOs) to an existing library object (refer to [StorePipeline](#)).
- Serializing a PSO library into a contiguous block of memory for disk storage (refer to [Serialize](#)).

- De-serializing a PSO library from persistent storage (this is handled by [CreatePipelineLibrary](#)).
- Retrieving individual PSOs from the library (refer to [LoadComputePipeline](#) and [LoadGraphicsPipeline](#)).

At no point in the lifecycle of a pipeline library is there duplication between PSOs with identical sub-components.

A recommended solution for managing the lifetime of the provided pointer while only having to ref-count the returned interface is to leverage [ID3D12Object::SetPrivateDataInterface](#), and use an object which implements [IUnknown](#), and frees the memory when the ref-count reaches 0.

Thread Safety

The pipeline library is thread-safe to use, and will internally synchronize as necessary, with one exception: multiple threads loading the same PSO (via [LoadComputePipeline](#), [LoadGraphicsPipeline](#), or [LoadPipeline](#)) should synchronize themselves, as this act may modify the state of that pipeline within the library in a non-thread-safe manner.

Examples

See the [Direct3D 12 pipeline state cache sample](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

- [ID3D12Device1](#)
- [Direct3D 12 pipeline state cache sample](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device1::SetEventOnMultipleFenceCompletion method (d3d12.h)

Article02/22/2024

Specifies an event that should be fired when one or more of a collection of fences reach specific values.

Syntax

C++

```
HRESULT SetEventOnMultipleFenceCompletion(
    [in] ID3D12Fence* ppFences,
    [in] const UINT64* pFenceValues,
    UINT NumFences,
    D3D12_MULTIPLE_FENCE_WAIT_FLAGS Flags,
    HANDLE hEvent
);
```

Parameters

[in] ppFences

Type: [ID3D12Fence*](#)

An array of length *NumFences* that specifies the [ID3D12Fence](#) objects.

[in] pFenceValues

Type: [const UINT64*](#)

An array of length *NumFences* that specifies the fence values required for the event is to be signaled.

NumFences

Type: [UINT](#)

Specifies the number of fences to be included.

Flags

Type: [D3D12_MULTIPLE_FENCE_WAIT_FLAGS](#)

Specifies one of the [D3D12_MULTIPLE_FENCE_WAIT_FLAGS](#) that determines how to proceed.

`hEvent`

Type: [HANDLE](#)

A handle to the event object.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code.

Remarks

To specify a single fence refer to the [SetEventOnCompletion](#) method.

If *hEvent* is a null handle, then this API will not return until the specified fence value(s) have been reached.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device1::SetResidencyPriority method (d3d12.h)

Article 02/22/2024

This method sets residency priorities of a specified list of objects.

Syntax

C++

```
HRESULT SetResidencyPriority(
    UINT NumObjects,
    [in] ID3D12Pageable *ppObjects,
    [in] const D3D12_RESIDENCY_PRIORITY *pPriorities
);
```

Parameters

NumObjects

Type: **UINT**

Specifies the number of objects in the *ppObjects* and *pPriorities* arrays.

[in] ppObjects

Type: **ID3D12Pageable***

Specifies an array, of length *NumObjects*, containing references to **ID3D12Pageable** objects.

[in] pPriorities

Type: **const D3D12_RESIDENCY_PRIORITY***

Specifies an array, of length *NumObjects*, of **D3D12_RESIDENCY_PRIORITY** values for the list of objects.

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code.

Remarks

For more information, refer to [Residency](#).

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device10 interface (d3d12.h)

Article02/22/2024

Represents a virtual adapter.

Requires the DirectX 12 Agility SDK 1.7 or later.

Inheritance

The [ID3D12Device10](#) interface derives from the [ID3D12Device9](#) interface.

Methods

The [ID3D12Device10](#) interface has these methods.

[+] Expand table

ID3D12Device10::CreateCommittedResource3
Creates a committed resource with an initial layout rather than an initial state.
ID3D12Device10::CreatePlacedResource2
Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.
ID3D12Device10::CreateReservedResource2
Creates a resource that is reserved, and not yet mapped to any pages in a heap.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device10::CreateCommittedResource3 method (d3d12.h)

Article11/03/2022

Creates a committed resource with an initial layout rather than an initial state.

Requires the DirectX 12 Agility SDK 1.7 or later.

Syntax

C++

```
HRESULT CreateCommittedResource3(
    const D3D12_HEAP_PROPERTIES      *pHeapProperties,
    D3D12_HEAP_FLAGS                HeapFlags,
    const D3D12_RESOURCE_DESC1       *pDesc,
    D3D12_BARRIER_LAYOUT            InitialLayout,
    const D3D12_CLEAR_VALUE          *pOptimizedClearValue,
    ID3D12ProtectedResourceSession  *pProtectedSession,
    UINT32                           NumCastableFormats,
    const DXGI_FORMAT                *pCastableFormats,
    REFIID                          riidResource,
    void                            **ppvResource
);
```

Parameters

pHeapProperties

Type: `_In_ const D3D12_HEAP_PROPERTIES*`

A pointer to a `D3D12_HEAP_PROPERTIES` structure that provides properties for the resource's heap.

HeapFlags

Type: `D3D12_HEAP_FLAGS`

Heap options, as a bitwise-OR'd combination of `D3D12_HEAP_FLAGS` enumeration constants.

pDesc

Type: **const D3D12_RESOURCE_DESC1***

A pointer to a **D3D12_RESOURCE_DESC1** structure that describes the resource, including a mip region.

InitialLayout

The initial layout of the texture resource;

D3D12_BARRIER_LAYOUT::D3D12_BARRIER_LAYOUT_UNDEFINED for buffers.

pOptimizedClearValue

Type: **const D3D12_CLEAR_VALUE***

Specifies a **D3D12_CLEAR_VALUE** structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal.

When the created resource is a texture with either the

D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET or

D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use **D3D12_RESOURCE_DIMENSION_BUFFER**, you must set

pOptimizedClearValue to `nullptr`.

pProtectedSession

Type: **ID3D12ProtectedResourceSession***

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the resource should be protected. You can obtain an **ID3D12ProtectedResourceSession** by calling **ID3D12Device4::CreateProtectedResourceSession**.

NumCastableFormats

The number of elements in *pCastableFormats*.

pCastableFormats

A contiguous array of **DXGI_FORMAT** structures that this resource can be cast to.

riidResource

Type: **REFIID**

A reference to the globally unique identifier (**GUID**) of the resource interface to return in *ppvResource*.

While *riidResource* is most commonly the **GUID** of [ID3D12Resource](#), it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

`ppvResource`

Type: **void****

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and [S_FALSE](#) is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

Requirement	Value
DLL	D3d12.dll

See also

[ID3D12Device10](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device10::CreatePlacedResource2 method (d3d12.h)

Article11/03/2022

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

Your application can re-use video memory by overlapping multiple Direct3D placed and reserved resources on heap regions. The simple memory re-use model (described in [Remarks](#)) exists to clarify which overlapping resource is valid at any given time. To maximize graphics tool support, with the simple model data-inheritance isn't supported; and finer-grained tile and sub-resource invalidation isn't supported. Only full overlapping resource invalidation occurs.

Requires the DirectX 12 Agility SDK 1.7 or later.

Syntax

C++

```
HRESULT CreatePlacedResource2(
    ID3D12Heap                 *pHeap,
    UINT64                      HeapOffset,
    const D3D12_RESOURCE_DESC1 *pDesc,
    D3D12_BARRIER_LAYOUT       InitialLayout,
    const D3D12_CLEAR_VALUE     *pOptimizedClearValue,
    UINT32                      NumCastableFormats,
    const DXGI_FORMAT           *pCastableFormats,
    REFIID                      riid,
    void                        **ppvResource
);
```

Parameters

pHeap

Type: [in] [ID3D12Heap*](#)

A pointer to the [ID3D12Heap](#) interface that represents the heap in which the resource is placed.

HeapOffset

Type: **UINT64**

The offset, in bytes, to the resource. The *HeapOffset* must be a multiple of the resource's alignment, and *HeapOffset* plus the resource size must be smaller than or equal to the heap size. [GetResourceAllocationInfo](#) must be used to understand the sizes of texture resources.

`pDesc`

Type: [in] const [D3D12_RESOURCE_DESC](#)*

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

`InitialLayout`

The initial layout of the texture resource;

[D3D12_BARRIER_LAYOUT::D3D12_BARRIER_LAYOUT_UNDEFINED](#) for buffers.

`pOptimizedClearValue`

Type: [in, optional] const [D3D12_CLEAR_VALUE](#)*

Specifies a [D3D12_CLEAR_VALUE](#) that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal.

When the created resource is a texture with either the

[D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or

[D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, your application should choose the value that the clear operation will most commonly be called with.

Clear operations can be called with other values, but those operations will not be as efficient as when the value matches the one passed into resource creation.

pOptimizedClearValue must be NULL when used with

[D3D12_RESOURCE_DIMENSION_BUFFER](#).

`NumCastableFormats`

The number of elements in *pCastableFormats*.

`pCastableFormats`

A contiguous array of [DXGI_FORMAT](#) structures that this resource can be cast to.

`riid`

Type: **REFIID**

The globally unique identifier (**GUID**) for the resource interface. This is an input parameter.

The **REFIID**, or **GUID**, of the interface to the resource can be obtained by using the `_uuidof` macro. For example, `_uuidof(ID3D12Resource)` gets the **GUID** of the interface to a resource. Although **riid** is, most commonly, the **GUID** for [ID3D12Resource](#), it may be any **GUID** for any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

`ppvResource`

Type: [out, optional] **void****

A pointer to a memory block that receives a pointer to the resource. *ppvResource* can be **NULL**, to enable capability testing. When *ppvResource* is **NULL**, no object will be created and **S_FALSE** will be returned when *pResourceDesc* and other parameters are valid.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the resource. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

See Remarks for [ID3D12Device::CreatePlacedResource](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CreateCommittedResource](#)

[CreateReservedResource](#)

[ID3D12Device10](#)

[Shared heaps](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device10::CreateReservedResource2 method (d3d12.h)

Article 11/03/2022

Creates a resource that is reserved, and not yet mapped to any pages in a heap.

Requires the DirectX 12 Agility SDK 1.7 or later.

Syntax

C++

```
HRESULT CreateReservedResource2(
    const D3D12_RESOURCE_DESC      *pDesc,
    D3D12_BARRIER_LAYOUT          InitialLayout,
    const D3D12_CLEAR_VALUE        *pOptimizedClearValue,
    ID3D12ProtectedResourceSession *pProtectedSession,
    UINT32                         NumCastableFormats,
    const DXGI_FORMAT              *pCastableFormats,
    [in] REFIID                      riid,
    void                           **ppvResource
);
```

Parameters

pDesc

Type: **const D3D12_RESOURCE_DESC***

A pointer to a **D3D12_RESOURCE_DESC** structure that describes the resource.

InitialLayout

The initial layout of the texture resource;

D3D12_BARRIER_LAYOUT::D3D12_BARRIER_LAYOUT_UNDEFINED for buffers.

pOptimizedClearValue

Type: **const D3D12_CLEAR_VALUE***

Specifies a **D3D12_CLEAR_VALUE** structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use [D3D12_RESOURCE_DIMENSION_BUFFER](#), you must set *pOptimizedClearValue* to `nullptr`.

`pProtectedSession`

Type: [ID3D12ProtectedResourceSession*](#)

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the resource should be protected. You can obtain an [ID3D12ProtectedResourceSession](#) by calling [ID3D12Device4::CreateProtectedResourceSession](#).

`NumCastableFormats`

The number of elements in *pCastableFormats*.

`pCastableFormats`

A contiguous array of [DXGI_FORMAT](#) structures that this resource can be cast to.

`[in] riid`

Type: [REFIID](#)

A reference to the globally unique identifier (**GUID**) of the resource interface to return in *ppvResource*. See [Remarks](#).

While *riidResource* is most commonly the **GUID** of [ID3D12Resource](#), it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

`ppvResource`

Type: [void**](#)

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and `S_FALSE` is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
<code>E_OUTOFMEMORY</code>	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

See Remarks for [ID3D12Device.CreateReservedResource](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	<code>d3d12.h</code>
Library	<code>D3d12.lib</code>
DLL	<code>D3d12.dll</code>

See also

[CreateCommittedResource](#)

[CreatePlacedResource](#)

[ID3D12Device10](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device2 interface (d3d12.h)

Article02/22/2024

Represents a virtual adapter. This interface extends [ID3D12Device1](#) to create pipeline state objects from pipeline state stream descriptions.

Note This interface was introduced in Windows 10 Creators Update. Applications targeting Windows 10 Creators Update should use this interface instead of earlier or later versions. Applications targeting an earlier or later version of Windows 10 should use the appropriate version of the [ID3D12Device](#) interface.

Inheritance

The [ID3D12Device2](#) interface inherits from [ID3D12Device1](#). [ID3D12Device2](#) also has these types of members:

Methods

The [ID3D12Device2](#) interface has these methods.

[+] [Expand table](#)

[ID3D12Device2::CreatePipelineState](#)

Creates a pipeline state object from a pipeline state stream description.

Remarks

Use [D3D12CreateDevice](#) to create a device.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

[ID3D12Device1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device2::CreatePipelineState method (d3d12.h)

Article 02/22/2024

Creates a pipeline state object from a pipeline state stream description.

Syntax

C++

```
HRESULT CreatePipelineState(
    const D3D12_PIPELINE_STATE_STREAM_DESC *pDesc,
    REFIID                         riid,
    [out] void                      **ppPipelineState
);
```

Parameters

pDesc

Type: **const D3D12_PIPELINE_STATE_STREAM_DESC***

The address of a **D3D12_PIPELINE_STATE_STREAM_DESC** structure that describes the pipeline state.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the pipeline state interface ([ID3D12PipelineState](#)).

The **REFIID**, or **GUID**, of the interface to the pipeline state can be obtained by using the **_uuidof()** macro. For example, **_uuidof(ID3D12PipelineState)** will get the **GUID** of the interface to a pipeline state.

[out] ppPipelineState

Type: **void****

SAL: *COM_Outptr*

A pointer to a memory block that receives a pointer to the [ID3D12PipelineState](#) interface for the pipeline state object.

The pipeline state object is an immutable state object. It contains no methods.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to create the pipeline state object. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

This function takes the pipeline description as a [D3D12_PIPELINE_STATE_STREAM_DESC](#) and combines the functionality of the [ID3D12Device::CreateGraphicsPipelineState](#) and [ID3D12Device::CreateComputePipelineState](#) functions, which take their pipeline description as the less-flexible [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structs, respectively.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

See [D3D12_PIPELINE_STATE_STREAM_DESC](#) for a description of the layout and behavior of a streaming pipeline desc.

[ID3D12Device2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device3 interface (d3d12.h)

Article02/22/2024

Represents a virtual adapter. This interface extends [ID3D12Device2](#) to support the creation of special-purpose diagnostic heaps in system memory that persist even in the event of a GPU-fault or device-removed scenario.

Note This interface, introduced in the Windows 10 Fall Creators Update, is the latest version of the [ID3D12Device](#) interface. Applications targeting the Windows 10 Fall Creators Update and later should use this interface instead of earlier versions.

Inheritance

The [ID3D12Device3](#) interface inherits from [ID3D12Device2](#). [ID3D12Device3](#) also has these types of members:

Methods

The [ID3D12Device3](#) interface has these methods.

[] [Expand table](#)

ID3D12Device3::EnqueueMakeResident
Asynchronously makes objects resident for the device.
ID3D12Device3::OpenExistingHeapFromAddress
Creates a special-purpose diagnostic heap in system memory from an address. The created heap can persist even in the event of a GPU-fault or device-removed scenario.
ID3D12Device3::OpenExistingHeapFromFileMapping
Creates a special-purpose diagnostic heap in system memory from a file mapping object. The created heap can persist even in the event of a GPU-fault or device-removed scenario.

Remarks

Use [D3D12CreateDevice](#) to create a device.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

[ID3D12Device1](#)

[ID3D12Device2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device3::EnqueueMakeResident method (d3d12.h)

Article 02/22/2024

Asynchronously makes objects resident for the device.

Syntax

C++

```
HRESULT EnqueueMakeResident(
    D3D12_RESIDENCY_FLAGS Flags,
    UINT NumObjects,
    [in] ID3D12Pageable* ppObjects,
    [in] ID3D12Fence* pFenceToSignal,
    UINT64 FenceValueToSignal
);
```

Parameters

Flags

Type: [D3D12_RESIDENCY_FLAGS](#)

Controls whether the objects should be made resident if the application is over its memory budget.

NumObjects

Type: [UINT](#)

The number of objects in the *ppObjects* array to make resident for the device.

[in] ppObjects

Type: [ID3D12Pageable*](#)

A pointer to a memory block; contains an array of [ID3D12Pageable](#) interface pointers for the objects.

Even though most D3D12 objects inherit from [ID3D12Pageable](#), residency changes are only supported on the following:

- descriptor heaps
- heaps
- committed resources
- query heaps

[in] pFenceToSignal

Type: [ID3D12Fence*](#)

A pointer to the fence used to signal when the work is done.

FenceValueToSignal

Type: [UINT64](#)

An unsigned 64-bit value signaled to the fence when the work is done.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

EnqueueMakeResident performs the same actions as [MakeResident](#), but does not wait for the resources to be made resident. Instead, **EnqueueMakeResident** signals a fence when the work is done.

The system will not allow work that references the resources that are being made resident by using **EnqueueMakeResident** before its fence is signaled. Instead, calls to this API are guaranteed to signal their corresponding fence in order, so the same fence can be used from call to call.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[ID3D12Device3](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device3::OpenExistingHeapFromAddress method (d3d12.h)

Article02/22/2024

Creates a special-purpose diagnostic heap in system memory from an address. The created heap can persist even in the event of a GPU-fault or device-removed scenario.

Syntax

C++

```
HRESULT OpenExistingHeapFromAddress(
    [in]  const void *pAddress,
    REFIID      riid,
    [out] void     **ppvHeap
);
```

Parameters

[in] pAddress

Type: **const void***

The address used to create the heap.

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the heap interface ([ID3D12Heap](#)).

The **REFIID**, or **GUID**, of the interface to the heap can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12Heap)` will retrieve the **GUID** of the interface to a heap.

[out] ppvHeap

Type: **void****

SAL: `COM_Outptr`

A pointer to a memory block. On success, the D3D12 runtime will write a pointer to the newly-opened heap into the memory block. The type of the pointer depends on the provided **riid** parameter.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to open the existing heap. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

The heap is created in system memory and permits CPU access. It wraps the entire VirtualAlloc region.

Hooks can be used for placed and reserved resources, as orthogonally as other heaps. Restrictions may still exist based on the flags that cannot be app-chosen.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device3 interface](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12Device3::OpenExistingHeapFromFileMapping method (d3d12.h)

Article 02/22/2024

Creates a special-purpose diagnostic heap in system memory from a file mapping object. The created heap can persist even in the event of a GPU-fault or device-removed scenario.

Syntax

C++

```
HRESULT OpenExistingHeapFromFileMapping(
    HANDLE hFileMapping,
    REFIID riid,
    [out] void    **ppvHeap
);
```

Parameters

`hFileMapping`

Type: [HANDLE](#)

The handle to the file mapping object to use to create the heap.

`riid`

Type: [REFIID](#)

The globally unique identifier ([GUID](#)) for the heap interface ([ID3D12Heap](#)).

The [REFIID](#), or [GUID](#), of the interface to the heap can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12Heap)` will retrieve the [GUID](#) of the interface to a heap.

`[out] ppvHeap`

Type: [void**](#)

SAL: `COM_Outptr`

A pointer to a memory block. On success, the D3D12 runtime will write a pointer to the newly-opened heap into the memory block. The type of the pointer depends on the provided **riid** parameter.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if there is insufficient memory to open the existing heap. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

The heap is created in system memory, and it permits CPU access. It wraps the entire VirtualAlloc region.

Hooks can be used for placed and reserved resources, as orthogonally as other heaps. Restrictions may still exist based on the flags that cannot be app-chosen.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device3 interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4 interface (d3d12.h)

Article06/12/2024

Represents a virtual adapter.

This interface extends [ID3D12Device3](#).

Inheritance

The **ID3D12Device4** interface inherits from the **ID3D12Device3** interface.

Methods

The **ID3D12Device4** interface has these methods.

[Expand table

ID3D12Device4::CreateCommandList1
Creates a command list in the closed state.
ID3D12Device4::CreateCommittedResource1
Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. (ID3D12Device4::CreateCommittedResource1)
ID3D12Device4::CreateHeap1
Creates a heap (optionally for a protected session) that can be used with placed resources and reserved resources.
ID3D12Device4::CreateProtectedResourceSession
Creates an object that represents a session for content protection.
ID3D12Device4::CreateReservedResource1
Creates a resource (optionally for a protected session) that is reserved, and not yet mapped to any pages in a heap.
ID3D12Device4::GetResourceAllocationInfo1

Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. (ID3D12Device4::GetResourceAllocationInfo1)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

Feedback

Was this page helpful?

[!\[\]\(4583127ea45a094ba8e651516834a671_img.jpg\) Yes](#)

[!\[\]\(9ef02707352e17c6c91de84896463a54_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4::CreateCommandList1 method (d3d12.h)

Article 02/22/2024

Creates a command list in the closed state. Also see [ID3D12Device::CreateCommandList](#).

Syntax

C++

```
HRESULT CreateCommandList1(
    [in]  UINT                  nodeMask,
    [in]  D3D12_COMMAND_LIST_TYPE type,
    [in]  D3D12_COMMAND_LIST_FLAGS flags,
    [in]  REFIID                riid,
    [out] void*                 **ppCommandList
);
```

Parameters

[in] nodeMask

Type: [UINT](#)

For single-GPU operation, set this to zero. If there are multiple GPU nodes, then set a bit to identify the node (the device's physical adapter) for which to create the command list. Each bit in the mask corresponds to a single node. Only one bit must be set. Also see [Multi-adapter systems](#).

[in] type

Type: [D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of command list to create.

flags

Type: [D3D12_COMMAND_LIST_FLAGS](#)

Specifies creation flags.

[in] riid

Type: **REFIID**

A reference to the globally unique identifier (**GUID**) of the command list interface to return in *ppCommandList*.

[out] *ppCommandList*

Type: **void****

A pointer to a memory block that receives a pointer to the [ID3D12CommandList](#) or [ID3D12GraphicsCommandList](#) interface for the command list.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the command list.

See [Direct3D 12 return codes](#) for other possible return values.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[ID3D12Device::CreateCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4::CreateCommittedResource1 method (d3d12.h)

Article 07/27/2022

Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. Also see [ID3D12Device::CreateCommittedResource](#) for a code example.

Syntax

C++

```
HRESULT CreateCommittedResource1(
    [in]           const D3D12_HEAP_PROPERTIES      *pHeapProperties,
    [in]           D3D12_HEAP_FLAGS                HeapFlags,
    [in]           const D3D12_RESOURCE_DESC        *pDesc,
    [in]           D3D12_RESOURCE_STATES         InitialResourceState,
    [in, optional] const D3D12_CLEAR_VALUE       *pOptimizedClearValue,
    [in, optional] ID3D12ProtectedResourceSession *pProtectedSession,
    [in]           REFIID                      riidResource,
    [out, optional] void*                      **ppvResource
);
```

Parameters

[in] pHeapProperties

Type: [const D3D12_HEAP_PROPERTIES*](#)

A pointer to a **D3D12_HEAP_PROPERTIES** structure that provides properties for the resource's heap.

[in] HeapFlags

Type: [D3D12_HEAP_FLAGS](#)

Heap options, as a bitwise-OR'd combination of **D3D12_HEAP_FLAGS** enumeration constants.

[in] pDesc

Type: [const D3D12_RESOURCE_DESC*](#)

A pointer to a **D3D12_RESOURCE_DESC** structure that describes the resource.

[in] InitialResourceState

Type: **D3D12_RESOURCE_STATES**

The initial state of the resource, as a bitwise-OR'd combination of **D3D12_RESOURCE_STATES** enumeration constants.

When you create a resource together with a **D3D12_HEAP_TYPE_UPLOAD** heap, you must set *InitialResourceState* to **D3D12_RESOURCE_STATE_GENERIC_READ**.

When you create a resource together with a **D3D12_HEAP_TYPE_READBACK** heap, you must set *InitialResourceState* to **D3D12_RESOURCE_STATE_COPY_DEST**.

[in, optional] pOptimizedClearValue

Type: **const D3D12_CLEAR_VALUE***

Specifies a **D3D12_CLEAR_VALUE** structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the **D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET** or **D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL** flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use **D3D12_RESOURCE_DIMENSION_BUFFER**, you must set *pOptimizedClearValue* to `nullptr`.

[in, optional] pProtectedSession

Type: **ID3D12ProtectedResourceSession***

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the resource should be protected. You can obtain an **ID3D12ProtectedResourceSession** by calling **ID3D12Device4::CreateProtectedResourceSession**.

[in] riidResource

Type: **REFIID**

A reference to the globally unique identifier (**GUID**) of the resource interface to return in *ppvResource*.

While *riidResource* is most commonly the **GUID** of [ID3D12Resource](#), it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

[out, optional] *ppvResource*

Type: **void****

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and [S_FALSE](#) is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

This method creates both a resource and a heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. The created heap is known as an implicit heap, because the heap object can't be obtained by the application. Before releasing the final reference on the resource, your application must ensure that the GPU will no longer read nor write to this resource.

The implicit heap is made resident for GPU access before the method returns control to your application. Also see [Residency](#).

The resource GPU VA mapping can't be changed. See [ID3D12CommandQueue::UpdateTileMappings](#) and [Volume tiled resources](#).

This method may be called by multiple threads concurrently.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4::CreateHeap1 method (d3d12.h)

Article12/15/2022

Creates a heap (optionally for a protected session) that can be used with placed resources and reserved resources. Also see [ID3D12Device::CreateHeap](#).

Syntax

C++

```
HRESULT CreateHeap1(
    [in]           const D3D12_HEAP_DESC          *pDesc,
    [in, optional] ID3D12ProtectedResourceSession *pProtectedSession,
    [in]           REFIID                      riid,
    [out, optional] void*                      **ppvHeap
);
```

Parameters

[in] pDesc

Type: [const D3D12_HEAP_DESC*](#)

A pointer to a constant **D3D12_HEAP_DESC** structure that describes the heap.

[in, optional] pProtectedSession

Type: [ID3D12ProtectedResourceSession*](#)

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the heap should be protected. You can obtain an **ID3D12ProtectedResourceSession** by calling [ID3D12Device4::CreateProtectedResourceSession](#).

A heap with a protected session can't be created with the [D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER](#) flag.

[in] riid

Type: **REFIID**

A reference to the globally unique identifier (**GUID**) of the heap interface to return in *ppvHeap*.

While *riidResource* is most commonly the **GUID** of [ID3D12Heap](#), it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

[out, optional] *ppvHeap*

Type: **void****

An optional pointer to a memory block that receives the requested interface pointer to the created heap object.

ppvHeap can be `nullptr`, to enable capability testing. When *ppvHeap* is `nullptr`, no object is created, and [S_FALSE](#) is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the heap.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

`CreateHeap1` creates a heap that can be used with placed resources and reserved resources.

Before releasing the final reference on the heap, your application must ensure that the GPU will no longer read or write to this heap.

A placed resource object holds a reference on the heap it is created on; but a reserved resource doesn't hold a reference for each mapping made to a heap.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4::CreateProtectedResourceSession method (d3d12.h)

Article 02/22/2024

Creates an object that represents a session for content protection. You can then provide that session when you're creating resource or heap objects, to indicate that they should be protected.

ⓘ Note

Memory contents can't be transferred from a protected resource to an unprotected resource.

Syntax

C++

```
HRESULT CreateProtectedResourceSession(
    [in]  const D3D12_PROTECTED_RESOURCE_SESSION_DESC *pDesc,
    [in]  REFIID                                     riid,
    [out] void*                                     ppSession
);
```

Parameters

[in] pDesc

Type: **const D3D12_PROTECTED_RESOURCE_SESSION_DESC***

A pointer to a constant **D3D12_PROTECTED_RESOURCE_SESSION_DESC** structure, describing the session to create.

[in] riid

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the **ID3D12ProtectedResourceSession** interface.

[out] ppSession

Type: **void****

A pointer to a memory block that receives an [ID3D12ProtectedResourceSession](#) interface pointer to the created session object.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device4::CreateReservedResource1 method (d3d12.h)

Article 10/20/2021

Creates a resource (optionally for a protected session) that is reserved, and not yet mapped to any pages in a heap. Also see [ID3D12Device::CreateReservedResource](#).

ⓘ Note

Only tiles from heaps created with the same protected resource session can be mapped into a protected reserved resource.

Syntax

C++

```
HRESULT CreateReservedResource1(
    [in]          const D3D12_RESOURCE_DESC      *pDesc,
    [in]          D3D12_RESOURCE_STATES        InitialState,
    [in, optional] const D3D12_CLEAR_VALUE     *pOptimizedClearValue,
    [in, optional] ID3D12ProtectedResourceSession *pProtectedSession,
    [in]          REFIID                      riid,
    [out, optional] void*                     **ppvResource
);
```

Parameters

[in] pDesc

Type: [const D3D12_RESOURCE_DESC*](#)

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

[in] InitialState

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

[in, optional] pOptimizedClearValue

Type: [const D3D12_CLEAR_VALUE*](#)

Specifies a [D3D12_CLEAR_VALUE](#) structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use [D3D12_RESOURCE_DIMENSION_BUFFER](#), you must set *pOptimizedClearValue* to `nullptr`.

`[in, optional] pProtectedSession`

Type: [ID3D12ProtectedResourceSession*](#)

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the resource should be protected. You can obtain an [ID3D12ProtectedResourceSession](#) by calling [ID3D12Device4::CreateProtectedResourceSession](#).

`[in] riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the resource interface to return in *ppvResource*. See [Remarks](#).

While *riidResource* is most commonly the GUID of [ID3D12Resource](#), it may be the GUID of any interface. If the resource object doesn't support the interface for this GUID, then creation fails with [E_NOINTERFACE](#).

`[out, optional] ppvResource`

Type: [void**](#)

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and [S_FALSE](#) is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

`CreateReservedResource` is equivalent to [D3D11_RESOURCE_MISC_TILED](#) in Direct3D 11. It creates a resource with virtual memory only, no backing store.

You need to map the resource to physical memory (that is, to a heap) using [CopyTileMappings](#) and [UpdateTileMappings](#).

These resource types can only be created when the adapter supports tiled resource tier 1 or greater. The tiled resource tier defines the behavior of accessing a resource that is not mapped to a heap.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[CreateCommittedResource1](#)

[CreatePlacedResource](#)

[ID3D12Device4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12Device4::GetResourceAllocationInfo1(UINT,UINT,const D3D12_RESOURCE_DESC*,D3D12_RESOURCE_ALLOCATION_INFO1*) method (d3d12.h)

Article 06/12/2024

Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. Also see [ID3D12Device::GetResourceAllocationInfo](#).

In addition to the [D3D12_RESOURCE_ALLOCATION_INFO](#) returned from the method, this version also returns an array of [D3D12_RESOURCE_ALLOCATION_INFO1](#) structures, which provide additional details for each resource description passed as input. See the *pResourceAllocationInfo1* parameter.

Syntax

C++

```
D3D12_RESOURCE_ALLOCATION_INFO GetResourceAllocationInfo1(
    [in]  UINT                      visibleMask,
    [in]  UINT                      numResourceDescs,
    [in]  const D3D12_RESOURCE_DESC*  *pResourceDescs,
    [out] D3D12_RESOURCE_ALLOCATION_INFO1* *pResourceAllocationInfo1
);
```

Parameters

[in] `visibleMask`

Type: [UINT](#)

For single-GPU operation, set this to zero. If there are multiple GPU nodes, then set bits to identify the nodes (the device's physical adapters). Each bit in the mask corresponds to a single node. Also see [Multi-adapter systems](#).

[in] `numResourceDescs`

Type: [UINT](#)

The number of resource descriptors in the *pResourceDescs* array. This is also the size (the number of elements in) *pResourceAllocationInfo1*.

[in] *pResourceDescs*

Type: **const D3D12_RESOURCE_DESC***

An array of **D3D12_RESOURCE_DESC** structures that described the resources to get info about.

[out] *pResourceAllocationInfo1*

Type: **D3D12_RESOURCE_ALLOCATION_INFO1***

An array of **D3D12_RESOURCE_ALLOCATION_INFO1** structures, containing additional details for each resource description passed as input. This makes it simpler for your application to allocate a heap for multiple resources, and without manually computing offsets for where each resource should be placed.

Return value

Type: **D3D12_RESOURCE_ALLOCATION_INFO**

A **D3D12_RESOURCE_ALLOCATION_INFO** structure that provides info about video memory allocated for the specified array of resources.

Remarks

When you're using [CreatePlacedResource](#), your application must use [GetResourceAllocationInfo](#) in order to understand the size and alignment characteristics of texture resources. The results of this method vary depending on the particular adapter, and must be treated as unique to this adapter and driver version.

Your application can't use the output of [GetResourceAllocationInfo](#) to understand packed mip properties of textures. To understand packed mip properties of textures, your application must use [GetResourceTiling](#).

Texture resource sizes significantly differ from the information returned by [GetResourceTiling](#), because some adapter architectures allocate extra memory for textures to reduce the effective bandwidth during common rendering scenarios. This even includes textures that have constraints on their texture layouts, or have standardized texture layouts. That extra memory can't be sparsely mapped nor

remapped by an application using [CreateReservedResource](#) and [UpdateTileMappings](#), so it isn't reported by [GetResourceTiling](#).

Your application can forgo using [GetResourceAllocationInfo](#) for buffer resources ([D3D12_RESOURCE_DIMENSION_BUFFER](#)). Buffers have the same size on all adapters, which is merely the smallest multiple of 64KB that's greater or equal to [D3D12_RESOURCE_DESC::Width](#).

When multiple resource descriptions are passed in, the C++ algorithm for calculating a structure size and alignment are used. For example, a three-element array with two tiny 64KB-aligned resources and a tiny 4MB-aligned resource, reports differing sizes based on the order of the array. If the 4MB aligned resource is in the middle, then the resulting **Size** is 12MB. Otherwise, the resulting **Size** is 8MB. The **Alignment** returned would always be 4MB, because it's the superset of all alignments in the resource array.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[ID3D12Device4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12Device5 interface (d3d12.h)

Article02/16/2023

Represents a virtual adapter.

This interface extends [ID3D12Device4](#).

ⓘ Note

This interface, introduced in Windows 10, version 1809, is the latest version of the [ID3D12Device](#) interface. Applications targeting Windows 10, version 1809 and later should use this interface instead of earlier versions.

Inheritance

The [ID3D12Device5](#) interface inherits from the [ID3D12Device4](#) interface.

Methods

The [ID3D12Device5](#) interface has these methods.

[+] [Expand table](#)

ID3D12Device5::CheckDriverMatchingIdentifier
Reports the compatibility of serialized data, such as a serialized raytracing acceleration structure resulting from a call to CopyRaytracingAccelerationStructure with mode <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE</code> , with the current device/driver.
ID3D12Device5::CreateLifetimeTracker
Creates a lifetime tracker associated with an application-defined callback; the callback receives notifications when the lifetime of a tracked object is changed.
ID3D12Device5::CreateMetaCommand
Creates an instance of the specified meta command.
ID3D12Device5::CreateStateObject

Creates an ID3D12StateObject .
ID3D12Device5::EnumerateMetaCommandParameters
Queries reflection metadata about the parameters of the specified meta command.
ID3D12Device5::EnumerateMetaCommands
Queries reflection metadata about available meta commands.
ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo
Query the driver for resource requirements to build an acceleration structure.
ID3D12Device5::RemoveDevice
You can call RemoveDevice to indicate to the Direct3D 12 runtime that the GPU device encountered a problem, and can no longer be used.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1809
Minimum supported server	Windows Server 2016
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::CheckDriverMatchingIdentifier method (d3d12.h)

Article 02/22/2024

Reports the compatibility of serialized data, such as a serialized raytracing acceleration structure resulting from a call to [CopyRaytracingAccelerationStructure](#) with mode [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE](#), with the current device/driver.

Syntax

C++

```
D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS CheckDriverMatchingIdentifier(
    [in] D3D12_SERIALIZED_DATA_TYPE
SerializedDataType,
    [in] const D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER
*pIdentifierToCheck
);
```

Parameters

[in] SerializedDataType

The type of the serialized data. For more information, see [D3D12_SERIALIZED_DATA_TYPE](#).

[in] pIdentifierToCheck

Identifier from the header of the serialized data to check with the driver. For more information, see [D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER](#).

Return value

The returned compatibility status. For more information, see [D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS](#).

Requirements

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::CreateLifetimeTracker method (d3d12.h)

Article02/22/2024

Creates a lifetime tracker associated with an application-defined callback; the callback receives notifications when the lifetime of a tracked object is changed.

Syntax

C++

```
HRESULT CreateLifetimeTracker(
    [in] ID3D12LifetimeOwner *pOwner,
    [in] REFIID             riid,
    [out] void              **ppvTracker
);
```

Parameters

[in] pOwner

Type: [ID3D12LifetimeOwner*](#)

A pointer to an **ID3D12LifetimeOwner** interface representing the application-defined callback.

[in] riid

Type: [REFIID](#)

A reference to the interface identifier (IID) of the interface to return in *ppvTracker*.

[out] ppvTracker

Type: [void**](#)

A pointer to a memory block that receives the requested interface pointer to the created object.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::CreateMetaCommand method (d3d12.h)

Article02/22/2024

Creates an instance of the specified meta command.

Syntax

C++

```
HRESULT CreateMetaCommand(
    [in]          REFGUID    CommandId,
    [in]          UINT       NodeMask,
    [in, optional] const void *pCreationParametersData,
    [in]          SIZE_T     CreationParametersDataSizeInBytes,
                           REFIID     riid,
    [out]         void       **ppMetaCommand
);
```

Parameters

[in] CommandId

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the meta command that you wish to instantiate.

[in] NodeMask

Type: **UINT**

For single-adapter operation, set this to zero. If there are multiple adapter nodes, set a bit to identify the node (one of the device's physical adapters) to which the meta command applies. Each bit in the mask corresponds to a single node. Only one bit must be set. See [Multi-adapter systems](#).

[in, optional] pCreationParametersData

Type: **const void***

An optional pointer to a constant structure containing the values of the parameters for creating the meta command.

[in] `CreationParametersDataSizeInBytes`

Type: `SIZE_T`

A `SIZE_T` containing the size of the structure pointed to by `pCreationParametersData`, if set, otherwise 0.

`riid`

Type: `REFIID`

A reference to the globally unique identifier (GUID) of the interface that you wish to be returned in `ppMetaCommand`. This is expected to be the GUID of [ID3D12MetaCommand](#).

[out] `ppMetaCommand`

Type: `void**`

A pointer to a memory block that receives a pointer to the meta command. This is the address of a pointer to an [ID3D12MetaCommand](#), representing the meta command created.

Return value

Type: `HRESULT`

If this method succeeds, it returns `S_OK`. Otherwise, it returns an `HRESULT` error code.

[+] Expand table

Return value	Description
<code>DXGI_ERROR_UNSUPPORTED</code>	The current hardware does not support the algorithm being requested

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::CreateStateObject method (d3d12.h)

Article 02/22/2024

Creates an [ID3D12StateObject](#).

Syntax

C++

```
HRESULT CreateStateObject(
    [in]  const D3D12_STATE_OBJECT_DESC *pDesc,
          REFIID                      riid,
    [out] void           **ppStateObject
);
```

Parameters

[in] pDesc

The description of the state object to create.

riid

The GUID of the interface to create. Use `_uuidof(ID3D12StateObject)`.

[out] ppStateObject

The returned state object.

Return value

Returns `S_OK` if successful; otherwise, returns one of the following values:

- `E_INVALIDARG` if one of the input parameters is invalid.
- `E_OUTOFMEMORY` if sufficient memory is not available to create the handle.
- Possibly other error codes that are described in the [Direct3D 12 Return Codes](#) topic.

Requirements

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::EnumerateMetaCommandParameters method (d3d12.h)

Article 02/22/2024

Queries reflection metadata about the parameters of the specified meta command.

Syntax

C++

```
HRESULT EnumerateMetaCommandParameters(
    [in]           REFGUID             CommandId,
    [in]           D3D12_META_COMMAND_PARAMETER_STAGE Stage,
    [out, optional] UINT              *pTotalStructureSizeInBytes,
    [in, out]       UINT              *pParameterCount,
    [out, optional] D3D12_META_COMMAND_PARAMETER_DESC *pParameterDescs
);
```

Parameters

[in] CommandId

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the meta command whose parameters you wish to be returned in *pParameterDescs*.

[in] Stage

Type: **D3D12_META_COMMAND_PARAMETER_STAGE**

A **D3D12_META_COMMAND_PARAMETER_STAGE** specifying the stage of the parameters that you wish to be included in the query.

[out, optional] pTotalStructureSizeInBytes

Type: **UINT***

An optional pointer to a **UINT** containing the size of the structure containing the parameter values, which you pass when creating/initializing/executing the meta command, as appropriate.

[in, out] pParameterCount

Type: **UINT***

A pointer to a **UINT** containing the number of parameters to query for. This field determines the size of the *pParameterDescs* array, unless *pParameterDescs* is **nullptr**.

[out, optional] pParameterDescs

Type: **D3D12_META_COMMAND_PARAMETER_DESC***

An optional pointer to an array of **D3D12_META_COMMAND_PARAMETER_DESC** containing the descriptions of the parameters. Pass **nullptr** to have the parameter count returned in *pParameterCount*.

Return value

Type: **HRESULT**

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::EnumerateMetaCommands method (d3d12.h)

Article 02/22/2024

Queries reflection metadata about available meta commands.

Syntax

C++

```
HRESULT EnumerateMetaCommands(
    UINT                 *pNumMetaCommands,
    D3D12_META_COMMAND_DESC *pDescs
);
```

Parameters

`pNumMetaCommands`

Type: [in, out] `UINT*`

A pointer to a `UINT` containing the number of meta commands to query for. This field determines the size of the `pDescs` array, unless `pDescs` is `nullptr`.

`pDescs`

Type: [out, optional] `D3D12_META_COMMAND_DESC*`

An optional pointer to an array of `D3D12_META_COMMAND_DESC` containing the descriptions of the available meta commands. Pass `nullptr` to have the number of available meta commands returned in `pNumMetaCommands`.

Return value

Type: `HRESULT`

If this method succeeds, it returns `S_OK`. Otherwise, it returns an `HRESULT` error code.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo method (d3d12.h)

Article 11/20/2024

Query the driver for resource requirements to build an acceleration structure.

Syntax

C++

```
void GetRaytracingAccelerationStructurePrebuildInfo(
    [in] const D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS *pDesc,
    [out] D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO     *pInfo
);
```

Parameters

[in] pDesc

Description of the acceleration structure build. This structure is shared with [BuildRaytracingAccelerationStructure](#). For more information, see [D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS](#).

The implementation is allowed to look at all the CPU parameters in this struct and nested structs. It may not inspect/dereference any GPU virtual addresses, other than to check to see if a pointer is NULL or not, such as the optional transform in [D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC](#), without dereferencing it. In other words, the calculation of resource requirements for the acceleration structure does not depend on the actual geometry data (such as vertex positions), rather it can only depend on overall properties, such as the number of triangles, number of instances etc.

[out] pInfo

The result of the query (in a [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO](#) structure).

Return value

None

Remarks

The input acceleration structure description is the same as what goes into [BuildRaytracingAccelerationStructure](#). The result of this function lets the application provide the correct amount of output storage and scratch storage to [BuildRaytracingAccelerationStructure](#) given the same geometry.

Builds can also be done with the same configuration passed to [GetAccelerationStructurePrebuildInfo](#) overall except equal or smaller counts for the number of geometries/instances or the number of vertices/indices/AABBS in any given geometry. In this case the storage requirements reported with the original sizes passed to [GetRaytracingAccelerationStructurePrebuildInfo](#) will be valid – the build may actually consume less space but not more. This is handy for app scenarios where having conservatively large storage allocated for acceleration structures is fine.

This method is on the device interface as opposed to command list on the assumption that drivers must be able to calculate resource requirements for an acceleration structure build from only looking at the CPU-visible portions of the call, without having to dereference any pointers to GPU memory containing actual vertex data, index data, etc.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device5::RemoveDevice method (d3d12.h)

Article 02/22/2024

You can call **RemoveDevice** to indicate to the Direct3D 12 runtime that the GPU device encountered a problem, and can no longer be used. Doing so will cause all devices' monitored fences to be signaled. Your application typically doesn't need to explicitly call **RemoveDevice**.

Syntax

C++

```
void RemoveDevice();
```

Return value

None

Remarks

Because device removal triggers all fences to be signaled to `UINT64_MAX`, you can create a callback for device removal using an event.

C++

```
HANDLE deviceRemovedEvent = CreateEventW(NULL, FALSE, FALSE, NULL);
assert(deviceRemovedEvent != NULL);
_deviceFence->SetEventOnCompletion(UINT64_MAX, deviceRemoved);

HANDLE waitHandle;
RegisterWaitForSingleObject(
    &waitHandle,
    deviceRemovedEvent,
    OnDeviceRemoved,
    _device.Get(), // Pass the device as our context
    INFINITE, // No timeout
    0 // No flags
);

void OnDeviceRemoved(PVOID context, BOOLEAN)
{
```

```
ID3D12Device* removedDevice = (ID3D12Device*)context;
HRESULT removedReason = removedDevice->GetDeviceRemovedReason();
// Perform app-specific device removed operation, such as logging or
inspecting DRED output
}
```

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device6 interface (d3d12.h)

Article02/22/2024

Represents a virtual adapter.

This interface extends [ID3D12Device5](#).

Inheritance

The **ID3D12Device6** interface inherits from the **ID3D12Device5** interface.

Methods

The **ID3D12Device6** interface has these methods.

[] [Expand table](#)

ID3D12Device6::SetBackgroundProcessingMode	Sets the mode for driver background processing optimizations.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device6::SetBackgroundProcessingMode method (d3d12.h)

Article 02/22/2024

Sets the mode for driver background processing optimizations.

Syntax

C++

```
HRESULT SetBackgroundProcessingMode(
    [in] D3D12_BACKGROUND_PROCESSING_MODE Mode,
    [in] D3D12_MEASUREMENTS_ACTION MeasurementsAction,
    [in] HANDLE hEventToSignalUponCompletion,
    [out] BOOL *pbFurtherMeasurementsDesired
);
```

Parameters

[in] Mode

Type: [D3D12_BACKGROUND_PROCESSING_MODE](#)

The level of dynamic optimization to apply to GPU work that's subsequently submitted.

[in] MeasurementsAction

Type: [D3D12_MEASUREMENTS_ACTION](#)

The action to take with the results of earlier workload instrumentation.

[in] hEventToSignalUponCompletion

Type: [HANDLE](#)

An optional handle to signal when the function is complete. For example, if *MeasurementsAction* is set to [D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS](#), then *hEventToSignalUponCompletion* is signaled when all resulting compilations have finished.

[out] pbFurtherMeasurementsDesired

Type: **BOOL***

An optional pointer to a Boolean value. The function sets the value to `true` to indicate that you should continue profiling, otherwise, `false`.

Remarks

A graphics driver can use idle-priority background CPU threads to dynamically recompile shader programs. That can improve GPU performance by specializing shader code to better match details of the hardware that it's running on, and/or the context in which it's being used.

As a developer, you don't have to do anything to benefit from this feature (over time, as drivers adopt background processing optimizations, existing shaders will automatically be tuned more efficiently). But, when you're profiling your code, you'll probably want to call **SetBackgroundProcessingMode** to make sure that any driver background processing optimizations have taken place before you take timing measurements. Here's an example.

C++

```
SetBackgroundProcessingMode(
    D3D12_BACKGROUND_PROCESSING_MODE_ALLOW_INTRUSIVE_MEASUREMENTS,
    D3D_MEASUREMENTS_ACTION_KEEP_ALL,
    nullptr, nullptr);

// Here, prime the system by rendering some typical content.
// For example, a level flythrough.

SetBackgroundProcessingMode(
    D3D12_BACKGROUND_PROCESSING_MODE_ALLOWED,
    D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS,
    nullptr, nullptr);

// Here, continue rendering. This time with dynamic optimizations applied.
// And then take your measurements.
```

PIX  automatically uses **SetBackgroundProcessingMode**—first to prime the system, and then to prevent any further changes from taking place in the middle of its analysis. PIX waits on an event (to make sure all background shader recompiles have finished) before it starts taking measurements.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device7 interface (d3d12.h)

Article02/16/2023

Represents a virtual adapter.

This interface extends [ID3D12Device6](#).

Inheritance

The **ID3D12Device7** interface inherits from the **ID3D12Device6** interface.

Methods

The **ID3D12Device7** interface has these methods.

[+] Expand table

ID3D12Device7::AddToStateObject	Incrementally add to an existing state object. This incurs lower CPU overhead than creating a state object from scratch that is a superset of an existing one.
ID3D12Device7::CreateProtectedResourceSession1	Revises the ID3D12Device4::CreateProtectedResourceSession method with provision GUID that indicates the type of protected resource session.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device7::AddToStateObject method (d3d12.h)

Article 02/22/2024

Incrementally add to an existing state object. This incurs lower CPU overhead than creating a state object from scratch that is a superset of an existing one (for example, adding a few more shaders).

Syntax

C++

```
HRESULT AddToStateObject(
    const D3D12_STATE_OBJECT_DESC *pAddition,
    ID3D12StateObject *pStateObjectToGrowFrom,
    REFIID riid,
    void **ppNewStateObject
);
```

Parameters

pAddition

Type: `_In_ const D3D12_STATE_OBJECT_DESC*`

Description of state object contents to add to existing state object. To help generate this see the `CD3D12_STATE_OBJECT_DESC` helper in class in `d3dx12.h`.

pStateObjectToGrowFrom

Type: `_In_ ID3D12StateObject*`

Existing state object, which can be in use (for example, active raytracing) during this operation.

The existing state object must not be of type `Collection`.

riid

Type: `_In_ REFIID`

Must be the IID of the `ID3D12StateObject` interface.

`ppNewStateObject`

Type: `_COM_Outptr_ void**`

Returned state object.

Behavior is undefined if shader identifiers are retrieved for new shaders from this call and they are accessed via shader tables by any already existing or in-flight command list that references some older state object. Use of the new shaders added to the state object can occur only from commands (such as `DispatchRays` or `ExecuteIndirect` calls) recorded in a command list after the call to `AddToStateObject`.

Return value

`S_OK` for success. `E_INVALIDARG`, `E_OUTOFMEMORY` on failure. The debug layer provides detailed status information.

Remarks

For more info, see [AddToStateObject](#).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

ID3D12Device7::CreateProtectedResourceSession1 method (d3d12.h)

Article 02/22/2024

`CreateProtectedResourceSession1` revises the [ID3D12Device4::CreateProtectedResourceSession](#) method with provision (in the structure passed via the *pDesc* parameter) for a globally unique identifier (**GUID**) that indicates the type of protected resource session.

Calling [ID3D12Device4::CreateProtectedResourceSession](#) is equivalent to calling `ID3D12Device7::CreateProtectedResourceSession1` with the `D3D12_PROTECTED_RESOURCES_SESSION_HARDWARE_PROTECTED` GUID.

Syntax

C++

```
HRESULT CreateProtectedResourceSession1(
    const D3D12_PROTECTED_RESOURCE_SESSION_DESC1 *pDesc,
    REFIID riid,
    void **ppSession
);
```

Parameters

`pDesc`

Type: `_In_ const D3D12_PROTECTED_RESOURCE_SESSION_DESC1*`

A pointer to a constant `D3D12_PROTECTED_RESOURCE_SESSION_DESC1` structure, describing the session to create.

`riid`

Type: `_In_ REFIID`

The GUID of the interface to a protected session. Most commonly, [ID3D12ProtectedResourceSession1](#), although it may be any **GUID** for any interface. If the protected session object doesn't support the interface for this **GUID**, the getter will return `E_NOINTERFACE`.

ppSession

Type: _COM_Outptr_ void**

A pointer to a memory block that receives a pointer to the session for the given protected session (the specific interface type returned depends on *riid*).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8 interface (d3d12.h)

Article06/12/2024

Represents a virtual adapter.

This interface extends [ID3D12Device7](#).

Inheritance

The **ID3D12Device8** interface inherits from the **ID3D12Device7** interface.

Methods

The **ID3D12Device8** interface has these methods.

[] [Expand table](#)

ID3D12Device8::CreateCommittedResource2
Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap.
ID3D12Device8::CreatePlacedResource1
Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.
ID3D12Device8::CreateSamplerFeedbackUnorderedAccessView
For purposes of sampler feedback, creates a descriptor suitable for binding.
ID3D12Device8::GetCopyableFootprints1
Gets a resource layout that can be copied. Helps your app fill in D3D12_PLACED_SUBRESOURCE_FOOTPRINT and D3D12_SUBRESOURCE_FOOTPRINT when suballocating space in upload heaps.
ID3D12Device8::GetResourceAllocationInfo2
Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. (ID3D12Device8::GetResourceAllocationInfo2)

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8::CreateCommittedResource2 method (d3d12.h)

Article 02/10/2023

Creates both a resource and an implicit heap (optionally for a protected session), such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. Also see [ID3D12Device::CreateCommittedResource](#) for a code example.

Syntax

C++

```
HRESULT CreateCommittedResource2(
    const D3D12_HEAP_PROPERTIES      *pHeapProperties,
    D3D12_HEAP_FLAGS                HeapFlags,
    const D3D12_RESOURCE_DESC1      *pDesc,
    D3D12_RESOURCE_STATES          InitialResourceState,
    const D3D12_CLEAR_VALUE         *pOptimizedClearValue,
    ID3D12ProtectedResourceSession *pProtectedSession,
    REFIID                          riidResource,
    void                            **ppvResource
);
```

Parameters

pHeapProperties

Type: `_In_ const D3D12_HEAP_PROPERTIES*`

A pointer to a `D3D12_HEAP_PROPERTIES` structure that provides properties for the resource's heap.

HeapFlags

Type: `D3D12_HEAP_FLAGS`

Heap options, as a bitwise-OR'd combination of `D3D12_HEAP_FLAGS` enumeration constants.

pDesc

Type: `const D3D12_RESOURCE_DESC1*`

A pointer to a [D3D12_RESOURCE_DESC1](#) structure that describes the resource, including a mip region.

`InitialResourceState`

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

When you create a resource together with a [D3D12_HEAP_TYPE_UPLOAD](#) heap, you must set *InitialResourceState* to [D3D12_RESOURCE_STATE_GENERIC_READ](#).

When you create a resource together with a [D3D12_HEAP_TYPE_READBACK](#) heap, you must set *InitialResourceState* to [D3D12_RESOURCE_STATE_COPY_DEST](#).

`pOptimizedClearValue`

Type: [const D3D12_CLEAR_VALUE*](#)

Specifies a [D3D12_CLEAR_VALUE](#) structure that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, you should choose the value with which the clear operation will most commonly be called. You can call the clear operation with other values, but those operations won't be as efficient as when the value matches the one passed in to resource creation.

When you use [D3D12_RESOURCE_DIMENSION_BUFFER](#), you must set *pOptimizedClearValue* to `nullptr`.

`pProtectedSession`

Type: [ID3D12ProtectedResourceSession*](#)

An optional pointer to an object that represents a session for content protection. If provided, this session indicates that the resource should be protected. You can obtain an [ID3D12ProtectedResourceSession](#) by calling [ID3D12Device4::CreateProtectedResourceSession](#).

`riidResource`

Type: [REFIID](#)

A reference to the globally unique identifier (**GUID**) of the resource interface to return in *ppvResource*.

While *riidResource* is most commonly the **GUID** of [ID3D12Resource](#), it may be the **GUID** of any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with [E_NOINTERFACE](#).

`ppvResource`

Type: **void****

An optional pointer to a memory block that receives the requested interface pointer to the created resource object.

ppvResource can be `nullptr`, to enable capability testing. When *ppvResource* is `nullptr`, no object is created, and [S_FALSE](#) is returned when *pDesc* is valid.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
E_OUTOFMEMORY	There is insufficient memory to create the resource.

See [Direct3D 12 return codes](#) for other possible return values.

Remarks

This method creates both a resource and a heap, such that the heap is big enough to contain the entire resource, and the resource is mapped to the heap. The created heap is known as an implicit heap, because the heap object can't be obtained by the application. Before releasing the final reference on the resource, your application must ensure that the GPU will no longer read nor write to this resource.

The implicit heap is made resident for GPU access before the method returns control to your application. Also see [Residency](#).

The resource GPU VA mapping can't be changed. See [ID3D12CommandQueue::UpdateTileMappings](#) and [Volume tiled resources](#).

This method may be called by multiple threads concurrently.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [Sampler feedback specification](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8::CreatePlacedResource1 method (d3d12.h)

Article02/10/2023

Creates a resource that is placed in a specific heap. Placed resources are the lightest weight resource objects available, and are the fastest to create and destroy.

Your application can re-use video memory by overlapping multiple Direct3D placed and reserved resources on heap regions. The simple memory re-use model (described in Remarks) exists to clarify which overlapping resource is valid at any given time. To maximize graphics tool support, with the simple model data-inheritance isn't supported; and finer-grained tile and sub-resource invalidation isn't supported. Only full overlapping resource invalidation occurs.

Syntax

C++

```
HRESULT CreatePlacedResource1(
    ID3D12Heap                 *pHeap,
    UINT64                      HeapOffset,
    const D3D12_RESOURCE_DESC1  *pDesc,
    D3D12_RESOURCE_STATES       InitialState,
    const D3D12_CLEAR_VALUE     *pOptimizedClearValue,
    REFIID                      riid,
    void                         **ppvResource
);
```

Parameters

pHeap

Type: [in] [ID3D12Heap*](#)

A pointer to the [ID3D12Heap](#) interface that represents the heap in which the resource is placed.

HeapOffset

Type: [UINT64](#)

The offset, in bytes, to the resource. The *HeapOffset* must be a multiple of the resource's alignment, and *HeapOffset* plus the resource size must be smaller than or equal to the heap size. [GetResourceAllocationInfo](#) must be used to understand the sizes of texture resources.

`pDesc`

Type: [in] const [D3D12_RESOURCE_DESC1](#)*

A pointer to a [D3D12_RESOURCE_DESC1](#) structure that describes the resource, including a mip region.

`InitialState`

Type: [D3D12_RESOURCE_STATES](#)

The initial state of the resource, as a bitwise-OR'd combination of [D3D12_RESOURCE_STATES](#) enumeration constants.

When a resource is created together with a [D3D12_HEAP_TYPE_UPLOAD](#) heap, *InitialState* must be [D3D12_RESOURCE_STATE_GENERIC_READ](#). When a resource is created together with a [D3D12_HEAP_TYPE_READBACK](#) heap, *InitialState* must be [D3D12_RESOURCE_STATE_COPY_DEST](#).

`pOptimizedClearValue`

Type: [in, optional] const [D3D12_CLEAR_VALUE](#)*

Specifies a [D3D12_CLEAR_VALUE](#) that describes the default value for a clear color.

pOptimizedClearValue specifies a value for which clear operations are most optimal. When the created resource is a texture with either the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) or [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flags, your application should choose the value that the clear operation will most commonly be called with.

Clear operations can be called with other values, but those operations will not be as efficient as when the value matches the one passed into resource creation.

pOptimizedClearValue must be NULL when used with [D3D12_RESOURCE_DIMENSION_BUFFER](#).

`riid`

Type: [REFIID](#)

The globally unique identifier (**GUID**) for the resource interface. This is an input parameter.

The **REFIID**, or **GUID**, of the interface to the resource can be obtained by using the `_uuidof` macro. For example, `_uuidof(ID3D12Resource)` gets the **GUID** of the interface to a resource. Although **riid** is, most commonly, the **GUID** for **ID3D12Resource**, it may be any **GUID** for any interface. If the resource object doesn't support the interface for this **GUID**, then creation fails with **E_NOINTERFACE**.

`ppvResource`

Type: [out, optional] **void****

A pointer to a memory block that receives a pointer to the resource. *ppvResource* can be **NULL**, to enable capability testing. When *ppvResource* is **NULL**, no object will be created and **S_FALSE** will be returned when *pResourceDesc* and other parameters are valid.

Return value

Type: **HRESULT**

This method returns **E_OUTOFMEMORY** if there is insufficient memory to create the resource. See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

See [ID3D12Device::CreatePlacedResource](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [CreateCommittedResource](#)
 - [CreateReservedResource](#)
 - [ID3D12Device](#)
 - [Shared heaps](#)
 - [Sampler feedback specification ↗](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8::CreateSamplerFeedbackUnorderedAccessView method (d3d12.h)

Article 02/22/2024

For purposes of sampler feedback, creates a descriptor suitable for binding.

Syntax

C++

```
void CreateSamplerFeedbackUnorderedAccessView(
    ID3D12Resource           *pTargetedResource,
    ID3D12Resource           *pFeedbackResource,
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor
);
```

Parameters

pTargetedResource

Type: `_In_opt_ ID3D12Resource*`

The targeted resource, such as a texture, to create a descriptor for.

pFeedbackResource

Type: `_In_opt_ ID3D12Resource*`

The feedback resource, such as a texture, to create a descriptor for.

DestDescriptor

Type: `_In_ D3D12_CPU_DESCRIPTOR_HANDLE`

The CPU descriptor handle.

Return value

None

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [Sampler feedback specification](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8::GetCopyableFootprints1 method (d3d12.h)

Article 10/20/2021

Gets a resource layout that can be copied. Helps your app fill in [D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#) and [D3D12_SUBRESOURCE_FOOTPRINT](#) when suballocating space in upload heaps.

Syntax

C++

```
void GetCopyableFootprints1(
    const D3D12_RESOURCE_DESC1* pResourceDesc,
    UINT FirstSubresource,
    UINT NumSubresources,
    UINT64 BaseOffset,
    D3D12_PLACED_SUBRESOURCE_FOOTPRINT* pLayouts,
    UINT* pNumRows,
    UINT64* pRowSizeInBytes,
    UINT64* pTotalBytes
);
```

Parameters

pResourceDesc

Type: [const D3D12_RESOURCE_DESC1*](#)

A description of the resource, as a pointer to a [D3D12_RESOURCE_DESC1](#) structure.

FirstSubresource

Type: [in] [UINT](#)

Index of the first subresource in the resource. The range of valid values is 0 to [D3D12_REQ_SUBRESOURCES](#).

NumSubresources

Type: [in] [UINT](#)

The number of subresources in the resource. The range of valid values is 0 to (`D3D12_REQ_SUBRESOURCES` - *FirstSubresource*).

`BaseOffset`

Type: `UINT64`

The offset, in bytes, to the resource.

`pLayouts`

Type: [out, optional] `D3D12_PLACED_SUBRESOURCE_FOOTPRINT*`

A pointer to an array (of length *NumSubresources*) of `D3D12_PLACED_SUBRESOURCE_FOOTPRINT` structures, to be filled with the description and placement of each subresource.

`pNumRows`

Type: [out, optional] `UINT*`

A pointer to an array (of length *NumSubresources*) of integer variables, to be filled with the number of rows for each subresource.

`pRowSizeInBytes`

Type: [out, optional] `UINT64*`

A pointer to an array (of length *NumSubresources*) of integer variables, each entry to be filled with the unpadded size in bytes of a row, of each subresource.

For example, if a Texture2D resource has a width of 32 and bytes per pixel of 4, then *pRowSizeInBytes* returns 128.

pRowSizeInBytes should not be confused with **row pitch**, as examining *pLayouts* and getting the row pitch from that will give you 256 as it is aligned to `D3D12_TEXTURE_DATA_PITCH_ALIGNMENT`.

`pTotalBytes`

Type: [out, optional] `UINT64*`

A pointer to an integer variable, to be filled with the total size, in bytes.

Return value

None

Remarks

For remarks and examples, see [ID3D12Device::GetCopyableFootprints](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [CD3DX12_RESOURCE_DESC](#)
- [CD3DX12_SUBRESOURCE_FOOTPRINT](#)
- [ID3D12Device](#)
- [Sampler feedback specification ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device8::GetResourceAllocationInfo2(UINT,UINT,const D3D12_RESOURCE_DESC1*,D3D12_RESOURCE_ALLOCATION_INFO1*) method (d3d12.h)

Article 06/12/2024

Gets rich info about the size and alignment of memory required for a collection of resources on this adapter. Also see [ID3D12Device4::GetResourceAllocationInfo1](#).

This version also returns an array of [D3D12_RESOURCE_DESC1](#) structures.

Syntax

C++

```
D3D12_RESOURCE_ALLOCATION_INFO GetResourceAllocationInfo2(
    UINT                      visibleMask,
    UINT                      numResourceDescs,
    const D3D12_RESOURCE_DESC1 *pResourceDescs,
    D3D12_RESOURCE_ALLOCATION_INFO1 *pResourceAllocationInfo1
);
```

Parameters

`visibleMask`

Type: [UINT](#)

For single-GPU operation, set this to zero. If there are multiple GPU nodes, then set bits to identify the nodes (the device's physical adapters). Each bit in the mask corresponds to a single node. Also see [Multi-adapter systems](#).

`numResourceDescs`

Type: [UINT](#)

The number of resource descriptors in the *pResourceDescs* array. This is also the size (the number of elements in) *pResourceAllocationInfo1*.

`pResourceDescs`

Type: [const D3D12_RESOURCE_DESC1*](#)

An array of [D3D12_RESOURCE_DESC1](#) structures that described the resources to get info about.

`pResourceAllocationInfo1`

Type: [D3D12_RESOURCE_ALLOCATION_INFO1*](#)

An array of [D3D12_RESOURCE_ALLOCATION_INFO1](#) structures, containing additional details for each resource description passed as input. This makes it simpler for your application to allocate a heap for multiple resources, and without manually computing offsets for where each resource should be placed.

Return value

Type: [D3D12_RESOURCE_ALLOCATION_INFO](#)

A [D3D12_RESOURCE_ALLOCATION_INFO](#) structure that provides info about video memory allocated for the specified array of resources.

Remarks

For remarks, see [ID3D12Device4::GetResourceAllocationInfo1](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- ID3D12Device8
 - Sampler feedback specification ↗
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device9 interface (d3d12.h)

Article 02/10/2023

Represents a virtual adapter. This interface extends [ID3D12Device8](#) to add methods to manage shader caches.

Inheritance

[ID3D12Device9](#) inherits from the [ID3D12Device8](#) interface.

Methods

The [ID3D12Device9](#) interface has these methods.

[+] Expand table

ID3D12Device9::CreateCommandQueue1
Creates a command queue with a creator ID.
ID3D12Device9::CreateShaderCacheSession
Creates an object that grants access to a shader cache, potentially opening an existing cache or creating a new one.
ID3D12Device9::ShaderCacheControl
Modifies the behavior of caches used internally by Direct3D or by the driver.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows

Requirement	Value
Header	d3d12.h

See also

- [ID3D12Device8](#)
 - [D3D12 shader cache APIs ↗](#)
-

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device9::CreateCommandQueue1 method (d3d12.h)

Article 02/22/2024

Creates a command queue with a creator ID.

Also see [ID3D12Device::CreateCommandQueue](#).

Syntax

C++

```
HRESULT CreateCommandQueue1(
    const D3D12_COMMAND_QUEUE_DESC *pDesc,
    REFIID CreatorID,
    REFIID riid,
    void **ppCommandQueue
);
```

Parameters

`pDesc`

Type: `_In_ const D3D12_COMMAND_QUEUE_DESC*`

Specifies a `D3D12_COMMAND_QUEUE_DESC` that describes the command queue.

`CreatorID`

Type: `REFIID`

A creator ID. See [Remarks](#).

`riid`

Type: `REFIID`

The globally unique identifier (GUID) for the command queue interface.

`ppCommandQueue`

Type: `_COM_Outptr_ void**`

A pointer to a memory block that receives a pointer to the [ID3D12CommandQueue](#) interface for the command queue.

Return value

Type: [HRESULT](#)

Returns [E_OUTOFMEMORY](#) if there's insufficient memory to create the command queue; otherwise [S_OK](#). See [Direct3D 12 return codes](#) for other possible return values.

Remarks

When multiple components in the same process are sharing a single Direct3D 12 device, often they will end up with separate workloads on independent command queues. In some hardware implementations, independent queues can run in parallel only with specific other command queues.

Direct3D 12 applies a first-come, first-serve grouping for queues, which might not work well for all application or component designs. To help inform Direct3D 12's grouping of queues, you can specify a *creator ID* (which is unique per component) that restricts the grouping to other queues with the same ID. When possible, a component should choose the same unique ID for all of its queues. Microsoft has reserved a few well-known creator IDs for use by Microsoft-developed implementations of APIs on top of Direct3D 12.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- ID3D12Device::CreateCommandQueue
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Device9::CreateShaderCacheSession method (d3d12.h)

Article 02/22/2024

Creates an object that grants access to a shader cache, potentially opening an existing cache or creating a new one.

Syntax

C++

```
HRESULT CreateShaderCacheSession(  
    const D3D12_SHADER_CACHE_SESSION_DESC *pDesc,  
    REFIID riid,  
    void **ppvSession  
) ;
```

Parameters

pDesc

Type: `_In_ const D3D12_SHADER_CACHE_SESSION_DESC*`

A `D3D12_SHADER_CACHE_SESSION_DESC` structure describing the shader cache session to create.

riid

Type: `REFIID`

The globally unique identifier (GUID) for the shader cache session interface.

ppvSession

Type: `_COM_Outptr_opt_ void**`

A pointer to a memory block that receives a pointer to the `ID3D12ShaderCacheSession` interface for the shader cache session.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

[Expand table](#)

Return value	Description
<code>DXGI_ERROR_ALREADY_EXISTS</code>	You tried to create a cache with an existing identifier. See D3D12_SHADER_CACHE_SESSION_DESC::Identifier .

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Device9::ShaderCacheControl method (d3d12.h)

Article 02/22/2024

Modifies the behavior of caches used internally by Direct3D or by the driver. **ShaderCacheControl** may be used only in developer mode.

Syntax

C++

```
HRESULT ShaderCacheControl(  
    D3D12_SHADER_CACHE_KIND_FLAGS    Kinds,  
    D3D12_SHADER_CACHE_CONTROL_FLAGS Control  
) ;
```

Parameters

Kinds

Type: [D3D12_SHADER_CACHE_KIND_FLAGS](#)

The caches to modify. Any one of these caches may or may not exist.

Control

Type: [D3D12_SHADER_CACHE_CONTROL_FLAGS](#)

The way in which to modify the caches. You can't pass both **DISABLE** and **ENABLE** at the same time; and you must pass at least one flag.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceChild interface (d3d12.h)

Article02/22/2024

An interface from which other core interfaces inherit from, including (but not limited to) [ID3D12PipelineLibrary](#), [ID3D12CommandList](#), [ID3D12Pageable](#), and [ID3D12RootSignature](#). It provides a method to get back to the device object it was created against.

Inheritance

The **ID3D12DeviceChild** interface inherits from [ID3D12Object](#). **ID3D12DeviceChild** also has these types of members:

Methods

The **ID3D12DeviceChild** interface has these methods.

[+] [Expand table](#)

<p>ID3D12DeviceChild::GetDevice</p> <p>Gets a pointer to the device that created this interface.</p>

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Object](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceChild::GetDevice method (d3d12.h)

Article 02/22/2024

Gets a pointer to the device that created this interface.

Syntax

C++

```
HRESULT GetDevice(
    REFIID riid,
    [out, optional] void    **ppvDevice
);
```

Parameters

riid

Type: **REFIID**

The globally unique identifier (**GUID**) for the device interface. The **REFIID**, or **GUID**, of the interface to the device can be obtained by using the `_uuidof()` macro. For example, `_uuidof(ID3D12Device)` will get the **GUID** of the interface to a device.

[out, optional] ppvDevice

Type: **void****

A pointer to a memory block that receives a pointer to the **ID3D12Device** interface for the device.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Any returned interfaces have their reference count incremented by one, so be sure to call `::release()` on the returned pointers before they are freed or else you will have a memory leak.

Examples

The [D3D12Multithreading](#) sample uses `ID3D12DeviceChild::GetDevice` as follows:

C++

```
// Returns required size of a buffer to be used for data upload
inline UINT64 GetRequiredIntermediateSize(
    _In_ ID3D12Resource* pDestinationResource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES) UINT FirstSubresource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES-FirstSubresource) UINT
NumSubresources)
{
    D3D12_RESOURCE_DESC Desc = pDestinationResource->GetDesc();
    UINT64 RequiredSize = 0;

    ID3D12Device* pDevice;
    pDestinationResource->GetDevice(__uuidof(*pDevice),
reinterpret_cast<void**>(&pDevice));
    pDevice->GetCopyableFootprints(&Desc, FirstSubresource, NumSubresources,
0, nullptr, nullptr, nullptr, &RequiredSize);
    pDevice->Release();

    return RequiredSize;
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData interface (d3d12.h)

Article 02/22/2024

Provides runtime access to Device Removed Extended Data (DRED) data. To retrieve the **ID3D12DeviceRemovedExtendedData** interface, call [QueryInterface](#) on an **ID3D12Device** (or derived) interface, passing the interface identifier (IID) of **ID3D12DeviceRemovedExtendedData**.

Inheritance

The **ID3D12DeviceRemovedExtendedData** interface inherits from the [IUnknown](#) interface.

Inheritance

The **ID3D12DeviceRemovedExtendedData** interface inherits from the [IUnknown](#) interface.

Methods

The **ID3D12DeviceRemovedExtendedData** interface has these methods.

[+] [Expand table](#)

[ID3D12DeviceRemovedExtendedData::GetAutoBreadcrumbsOutput](#)

Retrieves the Device Removed Extended Data (DRED) auto-breadcrumbs output after device removal.

[ID3D12DeviceRemovedExtendedData::GetPageFaultAllocationOutput](#)

Retrieves the Device Removed Extended Data (DRED) page fault data.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

- [Core interfaces](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

[!\[\]\(6d1bb32673c089f4b7f724ee7a352eff_img.jpg\) Yes](#)[!\[\]\(812b82343c12bccba081f99492fdb145_img.jpg\) No](#)

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData::GetAutoBreadcrumbsOutput method (d3d12.h)

Article 02/22/2024

Retrieves the Device Removed Extended Data (DRED) auto-breadcrumbs output after device removal.

Syntax

C++

```
HRESULT GetAutoBreadcrumbsOutput(  
    D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT *pOutput  
) ;
```

Parameters

pOutput

An output parameter that takes the address of a [D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT](#) object. The object whose address is passed receives the data.

Return value

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#). Returns [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#) if the device is *not* in a removed state. Returns [DXGI_ERROR_UNSUPPORTED](#) if auto-breadcrumbs have not been enabled with [ID3D12DeviceRemovedExtendedDataSettings::SetAutoBreadcrumbsEnablement](#).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedData interface](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData::GetPageFaultAllocationOutput method (d3d12.h)

Article02/22/2024

Retrieves the Device Removed Extended Data (DRED) page fault data, including matching allocation for both living and recently-deleted runtime objects. The object whose address is passed receives the data.

Syntax

C++

```
HRESULT GetPageFaultAllocationOutput(  
    D3D12_DRED_PAGE_FAULT_OUTPUT *pOutput  
) ;
```

Parameters

pOutput

An output parameter that takes the address of a [D3D12_DRED_PAGE_FAULT_OUTPUT](#) object.

Return value

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#). Returns [DXGI_ERROR_NOT_CURRENTLY_AVAILABLE](#) if the device is *not* in a removed state. Returns [DXGI_ERROR_UNSUPPORTED](#) if page fault reporting has not been enabled with [ID3D12DeviceRemovedExtendedDataSettings::SetPageFaultEnablement](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1903

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedData interface](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData1 interface (d3d12.h)

Article 02/22/2024

Inheritance

The **ID3D12DeviceRemovedExtendedData1** interface inherits from the **ID3D12DeviceRemovedExtendedData** interface.

Methods

The **ID3D12DeviceRemovedExtendedData1** interface has these methods.

[+] Expand table

ID3D12DeviceRemovedExtendedData1::GetAutoBreadcrumbsOutput1
ID3D12DeviceRemovedExtendedData1::GetPageFaultAllocationOutput1

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData1::GetAutoBreadcrumbsOutput1 method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetAutoBreadcrumbsOutput1(  
    D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT1 *pOutput  
);
```

Parameters

pOutput

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData1::GetPageFaultAllocationOutput1 method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetPageFaultAllocationOutput1(  
    D3D12_DRED_PAGE_FAULT_OUTPUT1 *pOutput  
);
```

Parameters

pOutput

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData2 interface (d3d12.h)

Article 02/22/2024

Inheritance

The **ID3D12DeviceRemovedExtendedData2** interface inherits from the **ID3D12DeviceRemovedExtendedData1** interface.

Methods

The **ID3D12DeviceRemovedExtendedData2** interface has these methods.

[+] Expand table

ID3D12DeviceRemovedExtendedData2::GetDeviceState
ID3D12DeviceRemovedExtendedData2::GetPageFaultAllocationOutput2

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData2::GetDeviceState method (d3d12.h)

Article02/22/2024

Syntax

C++

```
D3D12_DRED_DEVICE_STATE GetDeviceState();
```

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedData2::GetPageFaultAllocationOutput2 method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetPageFaultAllocationOutput2(
    D3D12_DRED_PAGE_FAULT_OUTPUT2 *pOutput
);
```

Parameters

pOutput

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings interface (d3d12.h)

Article02/22/2024

This interface controls Device Removed Extended Data (DRED) settings. You should configure all DRED settings before you create a Direct3D 12 device. To retrieve the **ID3D12DeviceRemovedExtendedDataSettings** interface, call [D3D12GetDebugInterface](#), passing the interface identifier (IID) of **ID3D12DeviceRemovedExtendedDataSettings**.

Inheritance

The **ID3D12DeviceRemovedExtendedDataSettings** interface inherits from the [IUnknown](#) interface.

Inheritance

The **ID3D12DeviceRemovedExtendedDataSettings** interface inherits from the [IUnknown](#) interface.

Methods

The **ID3D12DeviceRemovedExtendedDataSettings** interface has these methods.

[+] Expand table

ID3D12DeviceRemovedExtendedDataSettings::SetAutoBreadcrumbsEnablement
Configures the enablement settings for Device Removed Extended Data (DRED) auto-breadcrumbs.
ID3D12DeviceRemovedExtendedDataSettings::SetPageFaultEnablement
Configures the enablement settings for Device Removed Extended Data (DRED) page fault reporting.
ID3D12DeviceRemovedExtendedDataSettings::SetWatsonDumpEnablement
Configures the enablement settings for Device Removed Extended Data (DRED) Watson dump creation.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

- [Core interfaces](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings::SetAutoBreadcrumbsEnablement method (d3d12.h)

Article02/22/2024

Configures the enablement settings for Device Removed Extended Data (DRED) auto-breadcrumbs.

Syntax

C++

```
void SetAutoBreadcrumbsEnablement(  
    D3D12_DRED_ENABLEMENT Enablement  
);
```

Parameters

Enablement

A [D3D12_DRED_ENABLEMENT](#) value. The default is [D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED](#).

Return value

None

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings::SetPageFaultEnablement method (d3d12.h)

Article02/22/2024

Configures the enablement settings for Device Removed Extended Data (DRED) page fault reporting.

Syntax

C++

```
void SetPageFaultEnablement(  
    D3D12_DRED_ENABLEMENT Enablement  
);
```

Parameters

Enablement

A [D3D12_DRED_ENABLEMENT](#) value. The default is [D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED](#).

Return value

None

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings::SetWatsonDumpEnablement method (d3d12.h)

Article02/22/2024

Configures the enablement settings for Device Removed Extended Data (DRED) dump creation for [Windows Error Reporting \(WER\)](#), also known as Watson.

Syntax

C++

```
void SetWatsonDumpEnablement(  
    D3D12_DRED_ENABLEMENT Enablement  
);
```

Parameters

Enablement

A [D3D12_DRED_ENABLEMENT](#) value. The default is [D3D12_DRED_ENABLEMENT_SYSTEM_CONTROLLED](#).

Return value

None

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings1 interface (d3d12.h)

Article02/22/2024

Inheritance

The ID3D12DeviceRemovedExtendedDataSettings1 interface inherits from the ID3D12DeviceRemovedExtendedDataSettings interface.

Methods

The ID3D12DeviceRemovedExtendedDataSettings1 interface has these methods.

[+] Expand table

ID3D12DeviceRemovedExtendedDataSettings1::SetBreadcrumbContextEnablement
--

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DeviceRemovedExtendedDataSettings1::SetBreadcrumbContextEnablement method (d3d12.h)

Article02/22/2024

Syntax

C++

```
void SetBreadcrumbContextEnablement(
    D3D12_DRED_ENABLEMENT Enablement
);
```

Parameters

Enablement

Return value

None

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence interface (d3d12.h)

Article 07/27/2022

Represents a fence, an object used for synchronization of the CPU and one or more GPUs.

Inheritance

The **ID3D12Fence** interface inherits from [ID3D12Pageable](#). **ID3D12Fence** also has these types of members:

Methods

The **ID3D12Fence** interface has these methods.

[] Expand table

ID3D12Fence::GetCompletedValue	Gets the current value of the fence. (<code>ID3D12Fence.GetCompletedValue</code>)
ID3D12Fence::SetEventOnCompletion	Specifies an event that should be fired when the fence reaches a certain value. (<code>ID3D12Fence.SetEventOnCompletion</code>)
ID3D12Fence::Signal	Sets the fence to the specified value.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Fence Based Resource Management](#)

[ID3D12Pageable](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence::GetCompletedValue method (d3d12.h)

Article 02/22/2024

Gets the current value of the fence.

Syntax

C++

```
UINT64 GetCompletedValue();
```

Return value

Type: [UINT64](#)

Returns the current value of the fence. If the device has been removed, the return value will be [UINT64_MAX](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Fence](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence::SetEventOnCompletion method (d3d12.h)

Article02/22/2024

Specifies an event that's raised when the fence reaches a certain value.

Syntax

C++

```
HRESULT SetEventOnCompletion(
    UINT64 Value,
    HANDLE hEvent
);
```

Parameters

Value

Type: [UINT64](#)

The fence value when the event is to be signaled.

hEvent

Type: [HANDLE](#)

A handle to the event object.

Return value

Type: [HRESULT](#)

This method returns [E_OUTOFMEMORY](#) if the kernel components don't have sufficient memory to store the event in a list. See [Direct3D 12 return codes](#) for other possible return values.

Remarks

To specify multiple fences before an event is triggered, refer to [SetEventOnMultipleFenceCompletion](#).

If *hEvent* is a null handle, then this API will not return until the specified fence value(s) have been reached.

This method can be safely called from multiple threads at one time.

Examples

The [D3D12Multithreading](#) sample uses [ID3D12Fence::SetEventOnCompletion](#) as follows:

C++

```
// Wait for the command list to execute; we are reusing the same command
// list in our main loop but for now, we just want to wait for setup to
// complete before continuing.

// Signal and increment the fence value.
const UINT64 fenceToWaitFor = m_fenceValue;
ThrowIfFailed(m_commandQueue->Signal(m_fence.Get(), fenceToWaitFor));
m_fenceValue++;

// Wait until the fence is completed.
ThrowIfFailed(m_fence->SetEventOnCompletion(fenceToWaitFor, m_fenceEvent));
WaitForSingleObject(m_fenceEvent, INFINITE);
```

Refer to the [Example code in the Direct3D 12 reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence::Signal method (d3d12.h)

Article 02/22/2024

Sets the fence to the specified value.

Syntax

C++

```
HRESULT Signal(  
    UINT64 Value  
) ;
```

Parameters

Value

Type: [UINT64](#)

The value to set the fence to.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Use this method to set a fence value from the CPU side. Use [ID3D12CommandQueue::Signal](#) to set a fence from the GPU side.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Fence](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence1 interface (d3d12.h)

Article02/22/2024

Represents a fence. This interface extends [ID3D12Fence](#), and supports the retrieval of the flags used to create the original fence. This new feature is useful primarily for opening shared fences.

Note **ID3D12Fence1** was introduced in the Windows 10 Fall Creators Update, and is the latest version of the [ID3D12Fence](#) interface. Applications targeting Windows 10 Fall Creators Update and later should use **ID3D12Fence1** instead of earlier versions.

Inheritance

The **ID3D12Fence1** interface inherits from [ID3D12Fence](#). **ID3D12Fence1** also has these types of members:

Methods

The **ID3D12Fence1** interface has these methods.

[+] Expand table

[ID3D12Fence1::GetCreationFlags](#)

Gets the flags used to create the fence represented by the current instance.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Fence Based Resource Management](#)

[ID3D12fence](#)

[Multi-engine synchronization](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Fence1::GetCreationFlags method (d3d12.h)

Article 02/22/2024

Gets the flags used to create the fence represented by the current instance.

Syntax

C++

```
D3D12_FENCE_FLAGS GetCreationFlags();
```

Return value

Type: [D3D12_FENCE_FLAGS](#)

The flags used to create the fence.

Remarks

The flags returned by `GetCreationFlags` are used mainly for opening a shared fence.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Id3d12fence1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList interface (d3d12.h)

Article 07/27/2022

Encapsulates a list of graphics commands for rendering. Includes APIs for instrumenting the command list execution, and for setting and clearing the pipeline state.

Note The latest version of this interface is [ID3D12GraphicsCommandList1](#) introduced in the Windows 10 Creators Update. Applications targetting Windows 10 Creators Update should use the [ID3D12GraphicsCommandList1](#) interface instead of [ID3D12GraphicsCommandList](#).

Inheritance

The [ID3D12GraphicsCommandList](#) interface inherits from [ID3D12CommandList](#). [ID3D12GraphicsCommandList](#) also has these types of members:

Methods

The [ID3D12GraphicsCommandList](#) interface has these methods.

[] [Expand table](#)

ID3D12GraphicsCommandList::BeginEvent
Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (ID3D12GraphicsCommandList.BeginEvent)
ID3D12GraphicsCommandList::BeginQuery
Starts a query running. (ID3D12GraphicsCommandList.BeginQuery)
ID3D12GraphicsCommandList::ClearDepthStencilView
Clears the depth-stencil resource. (ID3D12GraphicsCommandList.ClearDepthStencilView)
ID3D12GraphicsCommandList::ClearRenderTargetView

Sets all the elements in a render target to one value.
ID3D12GraphicsCommandList::ClearState
Resets the state of a direct command list back to the state it was in when the command list was created. (ID3D12GraphicsCommandList.ClearState)
ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat
Sets all the elements in a unordered access view to the specified float values.
ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint
Sets all the elements in a unordered-access view (UAV) to the specified integer values.
ID3D12GraphicsCommandList::Close
Indicates that recording to the command list has finished. (ID3D12GraphicsCommandList.Close)
ID3D12GraphicsCommandList::CopyBufferRegion
Copies a region of a buffer from one resource to another.
ID3D12GraphicsCommandList::CopyResource
Copies the entire contents of the source resource to the destination resource.
ID3D12GraphicsCommandList::CopyTextureRegion
This method uses the GPU to copy texture data between two locations. Both the source and the destination may reference texture data located within either a buffer resource or a texture resource.
ID3D12GraphicsCommandList::CopyTiles
Copies tiles from buffer to tiled resource or vice versa. (ID3D12GraphicsCommandList.CopyTiles)
ID3D12GraphicsCommandList::DiscardResource
Discards a resource.
ID3D12GraphicsCommandList::Dispatch
Executes a compute shader on a thread group.
ID3D12GraphicsCommandList::DrawIndexedInstanced
Draws indexed, instanced primitives.

ID3D12GraphicsCommandList::DrawInstanced	Draws non-indexed, instanced primitives.
ID3D12GraphicsCommandList::EndEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (ID3D12GraphicsCommandList.EndEvent)
ID3D12GraphicsCommandList::EndQuery	Ends a running query.
ID3D12GraphicsCommandList::ExecuteBundle	Executes a bundle.
ID3D12GraphicsCommandList::ExecuteIndirect	Apps perform indirect draws/dispatches using the ExecuteIndirect method.
ID3D12GraphicsCommandList::IASetIndexBuffer	Sets the view for the index buffer.
ID3D12GraphicsCommandList::IASetPrimitiveTopology	Bind information about the primitive type, and data order that describes input data for the input assembler stage. (ID3D12GraphicsCommandList.IASetPrimitiveTopology)
ID3D12GraphicsCommandList::IASetVertexBuffers	Sets a CPU descriptor handle for the vertex buffers.
ID3D12GraphicsCommandList::OMSetBlendFactor	Sets the blend factor that modulate values for a pixel shader, render target, or both.
ID3D12GraphicsCommandList::OMSetRenderTargets	Sets CPU descriptor handles for the render targets and depth stencil.
ID3D12GraphicsCommandList::OMSetStencilRef	Sets the reference value for depth stencil tests.
ID3D12GraphicsCommandList::Reset	

Resets a command list back to its initial state as if a new command list was just created. (ID3D12GraphicsCommandList.Reset)
ID3D12GraphicsCommandList::ResolveQueryData
Extracts data from a query. ResolveQueryData works with all heap types (default, upload, and readback). ResolveQueryData works with all heap types (default, upload, and readback). .
ID3D12GraphicsCommandList::ResolveSubresource
Copy a multi-sampled resource into a non-multi-sampled resource.
ID3D12GraphicsCommandList::ResourceBarrier
Notifies the driver that it needs to synchronize multiple accesses to resources. (ID3D12GraphicsCommandList.ResourceBarrier)
ID3D12GraphicsCommandList::RSSetScissorRects
Binds an array of scissor rectangles to the rasterizer stage.
ID3D12GraphicsCommandList::RSSetViewports
Bind an array of viewports to the rasterizer stage of the pipeline. (ID3D12GraphicsCommandList.RSSetViewports)
ID3D12GraphicsCommandList::SetComputeRoot32BitConstant
Sets a constant in the compute root signature.
ID3D12GraphicsCommandList::SetComputeRoot32BitConstants
Sets a group of constants in the compute root signature.
ID3D12GraphicsCommandList::SetComputeRootConstantBufferView
Sets a CPU descriptor handle for the constant buffer in the compute root signature.
ID3D12GraphicsCommandList::SetComputeRootDescriptorTable
Sets a descriptor table into the compute root signature.
ID3D12GraphicsCommandList::SetComputeRootShaderResourceView
Sets a CPU descriptor handle for the shader resource in the compute root signature.
ID3D12GraphicsCommandList::SetComputeRootSignature
Sets the layout of the compute root signature.

[ID3D12GraphicsCommandList::SetComputeRootUnorderedAccessView](#)

Sets a CPU descriptor handle for the unordered-access-view resource in the compute root signature.

[ID3D12GraphicsCommandList::SetDescriptorHeaps](#)

Changes the currently bound descriptor heaps that are associated with a command list.

[ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant](#)

Sets a constant in the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants](#)

Sets a group of constants in the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView](#)

Sets a CPU descriptor handle for the constant buffer in the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable](#)

Sets a descriptor table into the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView](#)

Sets a CPU descriptor handle for the shader resource in the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRootSignature](#)

Sets the layout of the graphics root signature.

[ID3D12GraphicsCommandList::SetGraphicsRootUnorderedAccessView](#)

Sets a CPU descriptor handle for the unordered-access-view resource in the graphics root signature.

[ID3D12GraphicsCommandList::SetMarker](#)

Not intended to be called directly. Use the PIX event runtime to insert events into a command list. (ID3D12GraphicsCommandList::SetMarker)

[ID3D12GraphicsCommandList::SetPipelineState](#)

Sets all shaders and programs most of the fixed-function state of the graphics processing unit (GPU) pipeline.

[ID3D12GraphicsCommandList::SetPredication](#)

Sets a rendering predicate.

[ID3D12GraphicsCommandList::SOSetTargets](#)

Sets the stream output buffer views.

Remarks

This interface is new to D3D12, encapsulating much of the functionality of the [ID3D11CommandList](#) interface, and including the new functionality described in [Rendering](#).

Examples

The [D3D12nBodyGravity](#) sample uses [ID3D12GraphicsCommandList](#) as follows:

Declare the pipeline objects.

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

Populating command lists.

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12nBodyGravity::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
```

```

// fences to determine GPU execution progress.
ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

// However, when ExecuteCommandList() is called on a particular command
// list, that command list can then be reset at any time and must be
before
// re-recording.
ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

// Set necessary state.
m_commandList->SetPipelineState(m_pipelineState.Get());
m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

m_commandList->SetGraphicsRootConstantBufferView(RootParameterCB,
m_constantBufferGS->GetGPUVirtualAddress() + m_frameIndex *
sizeof(ConstantBufferGS));

ID3D12DescriptorHeap* ppHeaps[] = { m_srvUavHeap.Get() };
m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_POINTLIST);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

// Record commands.
const float clearColor[] = { 0.0f, 0.0f, 0.1f, 0.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);

// Render the particles.
float viewportHeight = static_cast<float>(static_cast<UINT>
(m_viewport.Height) / m_heightInstances);
float viewportWidth = static_cast<float>(static_cast<UINT>
(m_viewport.Width) / m_widthInstances);
for (UINT n = 0; n < ThreadCount; n++)
{
    const UINT srvIndex = n + (m_srvIndex[n] == 0 ? SrvParticlePosVelo0
: SrvParticlePosVelo1);

    D3D12_VIEWPORT viewport;
    viewport.TopLeftX = (n % m_widthInstances) * viewportWidth;
    viewport.TopLeftY = (n / m_widthInstances) * viewportHeight;
    viewport.Width = viewportWidth;
    viewport.Height = viewportHeight;
    viewport.MinDepth = D3D12_MIN_DEPTH;
    viewport.MaxDepth = D3D12_MAX_DEPTH;
}

```

```

    m_commandList->RSSetViewports(1, &viewport);

    CD3DX12_GPU_DESCRIPTOR_HANDLE srvHandle(m_srvUavHeap-
>GetGPUDescriptorHandleForHeapStart(), srvIndex, m_srvUavDescriptorSize);
    m_commandList->SetGraphicsRootDescriptorTable(RootParameterSRV,
srvHandle);

    m_commandList->DrawInstanced(ParticleCount, 1, 0, 0);
}

m_commandList->RSSetViewports(1, &m_viewport);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

ThrowIfFailed(m_commandList->Close());
}

```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12CommandList](#)

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

Yes

No

ID3D12GraphicsCommandList::BeginEvent method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command list.

Syntax

C++

```
void BeginEvent(
    UINT      Metadata,
    [in, optional] const void *pData,
    UINT      Size
);
```

Parameters

Metadata

Type: [UINT](#)

Internal.

[in, optional] pData

Type: [const void*](#)

Internal.

Size

Type: [UINT](#)

Internal.

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To mark the start of an instrumentation region at the current location within a D3D12 command list, use the **PIXBeginEvent** function or **PIXScopedEvent** macro. These are provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::BeginQuery method (d3d12.h)

Article 02/22/2024

Starts a query running.

Syntax

C++

```
void BeginQuery(  
    [in] ID3D12QueryHeap  *pQueryHeap,  
    [in] D3D12_QUERY_TYPE Type,  
    [in] UINT              Index  
);
```

Parameters

[in] pQueryHeap

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the query.

[in] Type

Type: [D3D12_QUERY_TYPE](#)

Specifies one member of [D3D12_QUERY_TYPE](#).

[in] Index

Type: [UINT](#)

Specifies the index of the query within the query heap.

Return value

None

Remarks

See [Queries](#) for more information about D3D12 queries.

Examples

The [D3D12PredicationQueries](#) sample uses `ID3D12GraphicsCommandList::BeginQuery` as follows:

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12PredicationQueries::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
before
    // re-recording.
    ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvHeap.Get() };
    m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList->ClearDepthStencilView(dsvHandle, D3D12_CLEAR_FLAG_DEPTH,
1.0f, 0, 0, nullptr);
```

```

// Draw the quads and perform the occlusion query.
{
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvFarQuad(m_cbvHeap-
>GetGPUDescriptorHandleForHeapStart(), m_frameIndex * CbvCountPerFrame,
m_cbvSrvDescriptorSize);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvNearQuad(cbvFarQuad,
m_cbvSrvDescriptorSize);

    m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);

        // Draw the far quad conditionally based on the result of the
occlusion query
        // from the previous frame.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPredication(m_queryResult.Get(), 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->DrawInstanced(4, 1, 0, 0);

        // Disable predication and always draw the near quad.
        m_commandList->SetPredication(nullptr, 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvNearQuad);
        m_commandList->DrawInstanced(4, 1, 4, 0);

        // Run the occlusion query with the bounding box quad.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPipelineState(m_queryState.Get());
        m_commandList->BeginQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);
        m_commandList->DrawInstanced(4, 1, 8, 0);
        m_commandList->EndQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);

        // Resolve the occlusion query and store the results in the query
result buffer
        // to be used on the subsequent frame.
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_PREDICATION, D3D12_RESOURCE_STATE_COPY_DEST));
        m_commandList->ResolveQueryData(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0, 1, m_queryResult.Get(), 0);
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_PREDICATION));
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

}

See Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ClearDepthStencilView method (d3d12.h)

Article 02/22/2024

Clears the depth-stencil resource.

Syntax

C++

```
void ClearDepthStencilView(
    [in] D3D12_CPU_DESCRIPTOR_HANDLE DepthStencilView,
    [in] D3D12_CLEAR_FLAGS           ClearFlags,
    [in] FLOAT                      Depth,
    [in] UINT8                     Stencil,
    [in] UINT                       NumRects,
    [in] const D3D12_RECT*          *pRects
);
```

Parameters

[in] DepthStencilView

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap for the depth stencil to be cleared.

[in] ClearFlags

Type: [D3D12_CLEAR_FLAGS](#)

A combination of [D3D12_CLEAR_FLAGS](#) values that are combined by using a bitwise OR operation. The resulting value identifies the type of data to clear (depth buffer, stencil buffer, or both).

[in] Depth

Type: [FLOAT](#)

A value to clear the depth buffer with. This value will be clamped between 0 and 1.

[in] Stencil

Type: **UINT8**

A value to clear the stencil buffer with.

[in] `NumRects`

Type: **UINT**

The number of rectangles in the array that the *pRects* parameter specifies.

[in] `pRects`

Type: **const D3D12_RECT***

An array of **D3D12_RECT** structures for the rectangles in the resource view to clear. If **NULL**, **ClearDepthStencilView** clears the entire resource view.

Return value

None

Remarks

Only direct and bundle command lists support this operation.

ClearDepthStencilView may be used to initialize resources which alias the same heap memory. See [CreatePlacedResource](#) for more details.

Runtime validation

For floating-point inputs, the runtime will set denormalized values to 0 (while preserving NaNs).

Validation failure will result in the call to [Close](#) returning **E_INVALIDARG**.

Debug layer

The debug layer will issue errors if the input colors are denormalized.

The debug layer will issue an error if the subresources referenced by the view are not in the appropriate state. For **ClearDepthStencilView**, the state must be in the state [D3D12_RESOURCE_STATE_DEPTH_WRITE](#).

Examples

The [D3D12Bundles](#) sample uses `ID3D12GraphicsCommandList::ClearDepthStencilView` as follows:

C++

```
// Pipeline objects.  
D3D12_VIEWPORT m_viewport;  
ComPtr<IDXGISwapChain3> m_swapChain;  
ComPtr<ID3D12Device> m_device;  
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];  
ComPtr<ID3D12Resource> m_depthStencil;  
ComPtr<ID3D12CommandAllocator> m_commandAllocator;  
ComPtr<ID3D12GraphicsCommandList> m_commandList;  
ComPtr<ID3D12CommandQueue> m_commandQueue;  
ComPtr<ID3D12RootSignature> m_rootSignature;  
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;  
ComPtr<ID3D12DescriptorHeap> m_cbvSrvHeap;  
ComPtr<ID3D12DescriptorHeap> m_dsvHeap;  
ComPtr<ID3D12DescriptorHeap> m_samplerHeap;  
ComPtr<ID3D12PipelineState> m_pipelineState1;  
ComPtr<ID3D12PipelineState> m_pipelineState2;  
D3D12_RECT m_scissorRect;
```

C++

```
void D3D12Bundles::PopulateCommandList(FrameResource* pFrameResource)  
{  
    // Command list allocators can only be reset when the associated  
    // command lists have finished execution on the GPU; apps should use  
    // fences to determine GPU execution progress.  
    ThrowIfFailed(m_pCurrentFrameResource->m_commandAllocator->Reset());  
  
    // However, when ExecuteCommandList() is called on a particular command  
    // list, that command list can then be reset at any time and must be  
before  
    // re-recording.  
    ThrowIfFailed(m_commandList->Reset(m_pCurrentFrameResource-  
        >m_commandAllocator.Get(), m_pipelineState1.Get()));  
  
    // Set necessary state.  
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());  
  
    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvSrvHeap.Get(),  
m_samplerHeap.Get() };  
    m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);  
  
    m_commandList->RSSetViewports(1, &m_viewport);  
    m_commandList->RSSetScissorRects(1, &m_scissorRect);
```

```

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList->ClearDepthStencilView(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart(), D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0,
nullptr);

    if (UseBundles)
    {
        // Execute the prebuilt bundle.
        m_commandList->ExecuteBundle(pFrameResource->m_bundle.Get());
    }
    else
    {
        // Populate a new command list.
        pFrameResource->PopulateCommandList(m_commandList.Get(),
m_pipelineState1.Get(), m_pipelineState2.Get(), m_currentFrameResourceIndex,
m_numIndices, &m_indexBufferView,
        &m_vertexBufferView, m_cbvSrvHeap.Get(), m_cbvSrvDescriptorSize,
m_samplerHeap.Get(), m_rootSignature.Get());
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

The [D3D12Multithreading](#) sample uses [ID3D12GraphicsCommandList::ClearDepthStencilView](#) as follows:

C++

```

void FrameResource::Init()
{
    // Reset the command allocators and lists for the main thread.
    for (int i = 0; i < CommandListCount; i++)
    {
        ThrowIfFailed(m_commandAllocators[i]->Reset());
        ThrowIfFailed(m_commandLists[i]->Reset(m_commandAllocators[i].Get(),

```

```

    m_pipelineState.Get());
}

// Clear the depth stencil buffer in preparation for rendering the
// shadow map.
m_commandLists[CommandListPre]->ClearDepthStencilView(m_shadowDepthView,
D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0, nullptr);

// Reset the worker command allocators and lists.
for (int i = 0; i < NumContexts; i++)
{
    ThrowIfFailed(m_shadowCommandAllocators[i]->Reset());
    ThrowIfFailed(m_shadowCommandLists[i]-
>Reset(m_shadowCommandAllocators[i].Get(), m_pipelineStateShadowMap.Get()));

    ThrowIfFailed(m_sceneCommandAllocators[i]->Reset());
    ThrowIfFailed(m_sceneCommandLists[i]-
>Reset(m_sceneCommandAllocators[i].Get(), m_pipelineState.Get()));
}
}

```

C++

```

// Assemble the CommandListPre command list.
void D3D12Multithreading::BeginFrame()
{
    m_pCurrentFrameResource->Init();

    // Indicate that the back buffer will be used as a render target.
    m_pCurrentFrameResource->m_commandLists[CommandListPre]-
>ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    // Clear the render target and depth stencil.
    const float clearColor[] = { 0.0f, 0.0f, 0.0f, 1.0f };
    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    m_pCurrentFrameResource->m_commandLists[CommandListPre]-
>ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_pCurrentFrameResource->m_commandLists[CommandListPre]-
>ClearDepthStencilView(m_dsvHeap->GetCPUDescriptorHandleForHeapStart(),
D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0, nullptr);

    ThrowIfFailed(m_pCurrentFrameResource->m_commandLists[CommandListPre]-
>Close());
}

// Assemble the CommandListMid command list.
void D3D12Multithreading::MidFrame()
{
    // Transition our shadow map from the shadow pass to readable in the
    // scene pass.
}
```

```
m_pCurrentFrameResource->SwapBarriers();  
  
ThrowIfFailed(m_pCurrentFrameResource->m_commandLists[CommandListMid]->Close());  
}
```

See Example code in the Direct3D 12 reference.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ClearRenderTargetView method (d3d12.h)

Article 10/13/2021

Sets all the elements in a render target to one value.

Syntax

C++

```
void ClearRenderTargetView(
    [in] D3D12_CPU_DESCRIPTOR_HANDLE RenderTargetView,
    [in] const FLOAT [4]                 ColorRGBA,
    [in] UINT           NumRects,
    [in] const D3D12_RECT*                *pRects
);
```

Parameters

[in] RenderTargetView

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Specifies a D3D12_CPU_DESCRIPTOR_HANDLE structure that describes the CPU descriptor handle that represents the start of the heap for the render target to be cleared.

[in] ColorRGBA

Type: **const FLOAT[4]**

A 4-component array that represents the color to fill the render target with.

[in] NumRects

Type: **UINT**

The number of rectangles in the array that the *pRects* parameter specifies.

[in] pRects

Type: **const D3D12_RECT***

An array of `D3D12_RECT` structures for the rectangles in the resource view to clear. If `NULL`, `ClearRenderTargetView` clears the entire resource view.

Return value

None

Remarks

`ClearRenderTargetView` may be used to initialize resources which alias the same heap memory. See [CreatePlacedResource](#) for more details.

Runtime validation

For floating-point inputs, the runtime will set denormalized values to 0 (while preserving NaNs).

Validation failure will result in the call to `Close` returning `E_INVALIDARG`.

Debug layer

The debug layer will issue errors if the input colors are denormalized.

The debug layer will issue an error if the subresources referenced by the view are not in the appropriate state. For `ClearRenderTargetView`, the state must be [D3D12_RESOURCE_STATE_RENDER_TARGET](#).

Examples

The [D3D12HelloTriangle](#) sample uses `ID3D12GraphicsCommandList::ClearRenderTargetView` as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
```

```
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
        // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
    m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
    >GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList-
    >IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
    m_commandList->DrawInstanced(3, 1, 0, 0);

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}
```

The D3D12Multithreading sample uses `ID3D12GraphicsCommandList::ClearRenderTargetView` as follows:

C++

```
// Frame resources.  
FrameResource* m_frameResources[FrameCount];  
FrameResource* m_pCurrentFrameResource;  
int m_currentFrameResourceIndex;
```

C++

```
// Assemble the CommandListPre command list.  
void D3D12Multithreading::BeginFrame()  
{  
    m_pCurrentFrameResource->Init();  
  
    // Indicate that the back buffer will be used as a render target.  
    m_pCurrentFrameResource->m_commandLists[CommandListPre]->ResourceBarrier(1,  
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),  
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));  
  
    // Clear the render target and depth stencil.  
    const float clearColor[] = { 0.0f, 0.0f, 0.0f, 1.0f };  
    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap->GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);  
    m_pCurrentFrameResource->m_commandLists[CommandListPre]->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);  
    m_pCurrentFrameResource->m_commandLists[CommandListPre]->ClearDepthStencilView(m_dsvHeap->GetCPUDescriptorHandleForHeapStart(),  
D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0, nullptr);  
  
    ThrowIfFailed(m_pCurrentFrameResource->m_commandLists[CommandListPre]->Close());  
}  
  
// Assemble the CommandListMid command list.  
void D3D12Multithreading::MidFrame()  
{  
    // Transition our shadow map from the shadow pass to readable in the  
    // scene pass.  
    m_pCurrentFrameResource->SwapBarriers();  
  
    ThrowIfFailed(m_pCurrentFrameResource->m_commandLists[CommandListMid]->Close());  
}
```

See Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::ClearState method (d3d12.h)

Article 02/22/2024

Resets the state of a direct command list back to the state it was in when the command list was created.

Syntax

C++

```
void ClearState(  
    [in, optional] ID3D12PipelineState *pPipelineState  
);
```

Parameters

[in, optional] pPipelineState

Type: [ID3D12PipelineState*](#)

A pointer to the [ID3D12PipelineState](#) object that contains the initial pipeline state for the command list.

Return value

None

Remarks

It is invalid to call **ClearState** on a bundle. If an app calls **ClearState** on a bundle, the call to [Close](#) will return [E_FAIL](#).

When **ClearState** is called, all currently bound resources are unbound. The primitive topology is set to [D3D_PRIMITIVE_TOPOLOGY_UNDEFINED](#). Viewports, scissor rectangles, stencil reference value, and the blend factor are set to empty values (all zeros). Predication is disabled.

The app-provided pipeline state object becomes bound as the currently set pipeline state object.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat method (d3d12.h)

Article 05/29/2024

Sets all of the elements in an unordered-access view (UAV) to the specified float values.

ⓘ Important

This behaves like a compute operation in that it isn't ordered with respect to surrounding work such as **Dispatch** calls. To ensure ordering, barrier calls must be issued before and/or after the **ClearUnorderedAccessViewXxx** call as needed. It might appear on some drivers that such barriers aren't necessary. But implicit barriers are not a spec guarantee; so they can't be relied upon. This is in contrast to **ClearDepthStencilView** and **ClearRenderTargetView** which (like **DrawXxx** commands), respect command list ordering.

Syntax

C++

```
void ClearUnorderedAccessViewFloat(
    D3D12_GPU_DESCRIPTOR_HANDLE ViewGPUHandleInCurrentHeap,
    D3D12_CPU_DESCRIPTOR_HANDLE ViewCPUHandle,
    ID3D12Resource* pResource,
    const FLOAT [4] Values,
    UINT NumRects,
    const D3D12_RECT* pRects
);
```

Parameters

`ViewGPUHandleInCurrentHeap`

Type: [in] [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A [D3D12_GPU_DESCRIPTOR_HANDLE](#) that references an initialized descriptor for the unordered-access view (UAV) that is to be cleared. This descriptor must be in a shader-visible descriptor heap, which must be set on the command list via [SetDescriptorHeaps](#).

`pViewCPUHandle`

Type: [in] [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A [D3D12_CPU_DESCRIPTOR_HANDLE](#) in a non-shader visible descriptor heap that references an initialized descriptor for the unordered-access view (UAV) that is to be cleared.

Important

This descriptor must not be in a shader-visible descriptor heap. This is to allow drivers that implement the clear as a fixed-function hardware operation (rather than as a dispatch) to efficiently read from the descriptor, as shader-visible heaps may be created in [WRITE_BACK](#) memory (similar to [D3D12_HEAP_TYPE_UPLOAD](#) heap types), and CPU reads from this type of memory are prohibitively slow.

`pResource`

Type: [in] [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the unordered-access-view (UAV) resource to clear.

`Values`

Type: [in] [const FLOAT\[4\]](#)

A 4-component array that containing the values to fill the unordered-access-view resource with.

`NumRects`

Type: [in] [UINT](#)

The number of rectangles in the array that the *pRects* parameter specifies.

`pRects`

Type: [in] [const D3D12_RECT*](#)

An array of [D3D12_RECT](#) structures for the rectangles in the resource view to clear. If `NULL`, [ClearUnorderedAccessViewFloat](#) clears the entire resource view.

Return value

None

Remarks

Runtime validation

For floating-point inputs, the runtime sets denormalized values to 0 (while preserving NaNs).

If you want to clear the UAV to a specific bit pattern, consider using [ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint](#).

Validation failure results in the call to [ID3D12GraphicsCommandList::Close](#) returning [E_INVALIDARG](#).

Debug layer

The debug layer issues errors if the input values are outside of a normalized range.

The debug layer issues an error if the subresources referenced by the view aren't in the appropriate state. For [ClearUnorderedAccessViewFloat](#), the state must be [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList interface](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint method (d3d12.h)

Article 05/29/2024

Sets all the elements in a unordered-access view (UAV) to the specified integer values.

ⓘ Important

This behaves like a compute operation in that it isn't ordered with respect to surrounding work such as **Dispatch** calls. To ensure ordering, barrier calls must be issued before and/or after the **ClearUnorderedAccessViewXxx** call as needed. It might appear on some drivers that such barriers aren't necessary. But implicit barriers are not a spec guarantee; so they can't be relied upon. This is in contrast to **ClearDepthStencilView** and **ClearRenderTargetView** which (like **DrawXxx** commands), respect command list ordering.

Syntax

C++

```
void ClearUnorderedAccessViewUint(
    D3D12_GPU_DESCRIPTOR_HANDLE ViewGPUHandleInCurrentHeap,
    D3D12_CPU_DESCRIPTOR_HANDLE ViewCPUHandle,
    ID3D12Resource* pResource,
    const UINT [4] Values,
    UINT NumRects,
    const D3D12_RECT* pRects
);
```

Parameters

`ViewGPUHandleInCurrentHeap`

Type: [in] [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A [D3D12_GPU_DESCRIPTOR_HANDLE](#) that references an initialized descriptor for the unordered-access view (UAV) that is to be cleared. This descriptor must be in a shader-visible descriptor heap, which must be set on the command list via [SetDescriptorHeaps](#).

`pViewCPUHandle`

Type: [in] [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A [D3D12_CPU_DESCRIPTOR_HANDLE](#) in a non-shader visible descriptor heap that references an initialized descriptor for the unordered-access view (UAV) that is to be cleared.

Important

This descriptor must not be in a shader-visible descriptor heap. This is to allow drivers that implement the clear as a fixed-function hardware operation (rather than as a dispatch) to efficiently read from the descriptor, as shader-visible heaps may be created in [WRITE_BACK](#) memory (similar to [D3D12_HEAP_TYPE_UPLOAD](#) heap types), and CPU reads from this type of memory are prohibitively slow.

`pResource`

Type: [in] [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the unordered-access-view (UAV) resource to clear.

`Values`

Type: [in] [const UINT\[4\]](#)

A 4-component array that containing the values to fill the unordered-access-view resource with.

`NumRects`

Type: [in] [UINT](#)

The number of rectangles in the array that the *pRects* parameter specifies.

`pRects`

Type: [in] [const D3D12_RECT*](#)

An array of [D3D12_RECT](#) structures for the rectangles in the resource view to clear. If `NULL`, [ClearUnorderedAccessViewUint](#) clears the entire resource view.

Return value

None

Remarks

Runtime validation

Validation failure results in the call to [ID3D12GraphicsCommandList::Close](#) returning `E_INVALIDARG`.

Debug layer

The debug layer issues errors if the input values are outside of a normalized range.

The debug layer issues an error if the subresources referenced by the view aren't in the appropriate state. For [ClearUnorderedAccessViewUint](#), the state must be `D3D12_RESOURCE_STATE_UNORDERED_ACCESS`.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList interface](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12GraphicsCommandList::Close method (d3d12.h)

Article 07/27/2022

Indicates that recording to the command list has finished.

Syntax

C++

```
HRESULT Close();
```

Return value

Type: [HRESULT](#)

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

- [E_FAIL](#) if the command list has already been closed, or an invalid API was called during command list recording.
- [E_OUTOFMEMORY](#) if the operating system ran out of memory during recording.
- [E_INVALIDARG](#) if an invalid argument was passed to the command list API during recording.

See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

The runtime will validate that the command list has not previously been closed. If an error was encountered during recording, the error code is returned here. The runtime won't call the close device driver interface (DDI) in this case.

Examples

The [D3D12HelloTriangle](#) sample uses [ID3D12GraphicsCommandList::Close](#) as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr m_swapChain;
ComPtr m_device;
ComPtr m_renderTargets[FrameCount];
ComPtr m_commandAllocator;
ComPtr m_commandQueue;
ComPtr m_rootSignature;
ComPtr m_rtvHeap;
ComPtr m_pipelineState;
ComPtr m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::LoadAssets()
{
    // Create an empty root signature.
    {
        CD3DX12_ROOT_SIGNATURE_DESC rootSignatureDesc;
        rootSignatureDesc.Init(0, nullptr, 0, nullptr,
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT);

        ComPtr signature;
        ComPtr error;
        ThrowIfFailed(D3D12SerializeRootSignature(&rootSignatureDesc,
D3D_ROOT_SIGNATURE_VERSION_1, &signature, &error));
        ThrowIfFailed(m_device->CreateRootSignature(0, signature-
>GetBufferPointer(), signature->GetBufferSize(),
IID_PPV_ARGS(&m_rootSignature)));
    }

    // Create the pipeline state, which includes compiling and loading
shaders.
    {
        ComPtr vertexShader;
        ComPtr pixelShader;

#if defined(_DEBUG)
        // Enable better shader debugging with the graphics debugging tools.
        UINT compileFlags = D3DCOMPILE_DEBUG | D3DCOMPILE_SKIP_OPTIMIZATION;
#else
        UINT compileFlags = 0;
#endif

        ThrowIfFailed(D3DCompileFromFile(GetAssetFullPath(L"shaders.hlsl").c_str(),
nullptr, nullptr, "VSMain", "vs_5_0", compileFlags, 0, &vertexShader,
nullptr));

        ThrowIfFailed(D3DCompileFromFile(GetAssetFullPath(L"shaders.hlsl").c_str(),
```

```

    nullptr, nullptr, "PSMain", "ps_5_0", compileFlags, 0, &pixelShader,
    nullptr));

        // Define the vertex input layout.
        D3D12_INPUT_ELEMENT_DESC inputElementDescs[] =
        {
            { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 },
                { "COLOR", 0, DXGI_FORMAT_R32G32B32A32_FLOAT, 0, 12,
D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA, 0 }
            };
        }

        // Describe and create the graphics pipeline state object (PSO).
        D3D12_GRAPHICS_PIPELINE_STATE_DESC psoDesc = {};
        psoDesc.InputLayout = { inputElementDescs,
_countof(inputElementDescs) };
        psoDesc.pRootSignature = m_rootSignature.Get();
        psoDesc.VS = { reinterpret_cast<UINT8*>(vertexShader-
>GetBufferPointer()), vertexShader->GetBufferSize() };
        psoDesc.PS = { reinterpret_cast<UINT8*>(pixelShader-
>GetBufferPointer()), pixelShader->GetBufferSize() };
        psoDesc.RasterizerState = CD3DX12_RASTERIZER_DESC(D3D12_DEFAULT);
        psoDesc.BlendState = CD3DX12_BLEND_DESC(D3D12_DEFAULT);
        psoDesc.DepthStencilState.DepthEnable = FALSE;
        psoDesc.DepthStencilState.StencilEnable = FALSE;
        psoDesc.SampleMask = UINT_MAX;
        psoDesc.PrimitiveTopologyType =
D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE;
        psoDesc.NumRenderTargets = 1;
        psoDesc.RTVFormats[0] = DXGI_FORMAT_R8G8B8A8_UNORM;
        psoDesc.SampleDesc.Count = 1;
        ThrowIfFailed(m_device->CreateGraphicsPipelineState(&psoDesc,
IID_PPV_ARGS(&m_pipelineState)));
    }

    // Create the command list.
    ThrowIfFailed(m_device->CreateCommandList(0,
D3D12_COMMAND_LIST_TYPE_DIRECT, m_commandAllocator.Get(),
m_pipelineState.Get(), IID_PPV_ARGS(&m_commandList)));

    // Command lists are created in the recording state, but there is
nothing
    // to record yet. The main loop expects it to be closed, so close it
now.
    ThrowIfFailed(m_commandList->Close());

    // Create the vertex buffer.
{
    // Define the geometry for a triangle.
    Vertex triangleVertices[] =
    {
        { { 0.0f, 0.25f * m_aspectRatio, 0.0f }, { 1.0f, 0.0f, 0.0f,
1.0f } },
            { { 0.25f, -0.25f * m_aspectRatio, 0.0f }, { 0.0f, 1.0f, 0.0f,
1.0f } },

```

```

        { { -0.25f, -0.25f * m_aspectRatio, 0.0f }, { 0.0f, 0.0f, 1.0f,
1.0f } }
    };

    const UINT vertexBufferSize = sizeof(triangleVertices);

    // Note: using upload heaps to transfer static data like vert
buffers is not
    // recommended. Every time the GPU needs it, the upload heap will be
marshalled
    // over. Please read up on Default Heap usage. An upload heap is
used here for
    // code simplicity and because there are very few verts to actually
transfer.

    ThrowIfFailed(m_device->CreateCommittedResource(
        &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),
        D3D12_HEAP_FLAG_NONE,
        &CD3DX12_RESOURCE_DESC::Buffer(vertexBufferSize),
        D3D12_RESOURCE_STATE_GENERIC_READ,
        nullptr,
        IID_PPV_ARGS(&m_vertexBuffer)));

    // Copy the triangle data to the vertex buffer.
    UINT8* pVertexDataBegin;
    CD3DX12_RANGE readRange(0, 0);           // We do not intend to read
from this resource on the CPU.
    ThrowIfFailed(m_vertexBuffer->Map(0, &readRange,
reinterpret_cast<void**>(&pVertexDataBegin)));
    memcpy(pVertexDataBegin, triangleVertices,
sizeof(triangleVertices));
    m_vertexBuffer->Unmap(0, nullptr);

    // Initialize the vertex buffer view.
    m_vertexBufferView.BufferLocation = m_vertexBuffer-
>GetGPUVirtualAddress();
    m_vertexBufferView.StrideInBytes = sizeof(Vertex);
    m_vertexBufferView.SizeInBytes = vertexBufferSize;
}

// Create synchronization objects and wait until assets have been
uploaded to the GPU.
{
    ThrowIfFailed(m_device->CreateFence(0, D3D12_FENCE_FLAG_NONE,
IID_PPV_ARGS(&m_fence)));
    m_fenceValue = 1;

    // Create an event handle to use for frame synchronization.
    m_fenceEvent = CreateEvent(nullptr, FALSE, FALSE, nullptr);
    if (m_fenceEvent == nullptr)
    {
        ThrowIfFailed(HRESULT_FROM_WIN32(GetLastError()));
    }

    // Wait for the command list to execute; we are reusing the same
command
}

```

```

        // list in our main loop but for now, we just want to wait for setup
        to
        // complete before continuing.
        WaitForPreviousFrame();
    }
}

```

C++

```

void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
    // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
    m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList-
    >IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
    m_commandList->DrawInstanced(3, 1, 0, 0);

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

}

See Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::CopyBufferRegion method (d3d12.h)

Article 02/22/2024

Copies a region of a buffer from one resource to another.

Syntax

C++

```
void CopyBufferRegion(  
    [in] ID3D12Resource *pDstBuffer,  
    UINT64 DstOffset,  
    [in] ID3D12Resource *pSrcBuffer,  
    UINT64 SrcOffset,  
    UINT64 NumBytes  
) ;
```

Parameters

[in] pDstBuffer

Type: [ID3D12Resource*](#)

Specifies the destination [ID3D12Resource](#).

DstOffset

Type: [UINT64](#)

Specifies a [UINT64](#) offset (in bytes) into the destination resource.

[in] pSrcBuffer

Type: [ID3D12Resource*](#)

Specifies the source [ID3D12Resource](#).

SrcOffset

Type: [UINT64](#)

Specifies a [UINT64](#) offset (in bytes) into the source resource, to start the copy from.

NumBytes

Type: **UINT64**

Specifies the number of bytes to copy.

Return value

None

Remarks

Consider using the [CopyResource](#) method when copying an entire resource, and use this method for copying regions of a resource.

[CopyBufferRegion](#) may be used to initialize resources which alias the same heap memory. See [CreatePlacedResource](#) for more details.

Examples

The [D3D12HelloTriangle](#) sample uses [ID3D12GraphicsCommandList::CopyBufferRegion](#) as follows:

C++

```
inline UINT64 UpdateSubresources(
    _In_ ID3D12GraphicsCommandList* pCmdList,
    _In_ ID3D12Resource* pDestinationResource,
    _In_ ID3D12Resource* pIntermediate,
    _In_range_(0,D3D12_REQ_SUBRESOURCES) UINT FirstSubresource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES-FirstSubresource) UINT
NumSubresources,
    UINT64 RequiredSize,
    _In_reads_(NumSubresources) const D3D12_PLACED_SUBRESOURCE_FOOTPRINT*
pLayouts,
    _In_reads_(NumSubresources) const UINT* pNumRows,
    _In_reads_(NumSubresources) const UINT64* pRowSizesInBytes,
    _In_reads_(NumSubresources) const D3D12_SUBRESOURCE_DATA* pSrcData)
{
    // Minor validation
    D3D12_RESOURCE_DESC IntermediateDesc = pIntermediate->GetDesc();
    D3D12_RESOURCE_DESC DestinationDesc = pDestinationResource->GetDesc();
    if (IntermediateDesc.Dimension != D3D12_RESOURCE_DIMENSION_BUFFER ||
        IntermediateDesc.Width < RequiredSize + pLayouts[0].Offset ||
        RequiredSize > (SIZE_T)-1 ||
        (DestinationDesc.Dimension == D3D12_RESOURCE_DIMENSION_BUFFER &&
        (FirstSubresource != 0 || NumSubresources != 1)))
}
```

```

    {
        return 0;
    }

    BYTE* pData;
    HRESULT hr = pIntermediate->Map(0, NULL, reinterpret_cast<void**>
(&pData));
    if (FAILED(hr))
    {
        return 0;
    }

    for (UINT i = 0; i < NumSubresources; ++i)
    {
        if (pRowSizesInBytes[i] > (SIZE_T)-1) return 0;
        D3D12_MEMCPY_DEST DestData = { pData + pLayouts[i].Offset,
pLayouts[i].Footprint.RowPitch, pLayouts[i].Footprint.RowPitch * pNumRows[i]
};
        MemcpySubresource(&DestData, &pSrcData[i],
(SIZE_T)pRowSizesInBytes[i], pNumRows[i], pLayouts[i].Footprint.Depth);
    }
    pIntermediate->Unmap(0, NULL);

    if (DestinationDesc.Dimension == D3D12_RESOURCE_DIMENSION_BUFFER)
    {
        CD3DX12_BOX SrcBox( UINT( pLayouts[0].Offset ), UINT(
pLayouts[0].Offset + pLayouts[0].Footprint.Width ) );
        pCmdList->CopyBufferRegion(
            pDestinationResource, 0, pIntermediate, pLayouts[0].Offset,
pLayouts[0].Footprint.Width);
    }
    else
    {
        for (UINT i = 0; i < NumSubresources; ++i)
        {
            CD3DX12_TEXTURE_COPY_LOCATION Dst(pDestinationResource, i +
FirstSubresource);
            CD3DX12_TEXTURE_COPY_LOCATION Src(pIntermediate, pLayouts[i]);
            pCmdList->CopyTextureRegion(&Dst, 0, 0, 0, &Src, nullptr);
        }
    }
    return RequiredSize;
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CopyTextureRegion](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::CopyResource method (d3d12.h)

Article 10/13/2021

Copies the entire contents of the source resource to the destination resource.

Syntax

C++

```
void CopyResource(  
    [in] ID3D12Resource *pDstResource,  
    [in] ID3D12Resource *pSrcResource  
) ;
```

Parameters

[in] pDstResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the destination resource.

[in] pSrcResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the source resource.

Return value

None

Remarks

CopyResource operations are performed on the GPU, and do not incur a significant CPU workload linearly dependent on the size of the data to copy.

CopyResource can be used to initialize resources that alias the same heap memory. See [CreatePlacedResource](#) for more details.

Debug layer

The debug layer issues an error if the source subresource is not in the [D3D12_RESOURCE_STATE_COPY_SOURCE](#) state.

The debug layer issues an error if the destination subresource is not in the [D3D12_RESOURCE_STATE_COPY_DEST](#) state.

Restrictions

This method has a few restrictions designed for improving performance. For instance, the source and destination resources:

- Must be different resources.
- Must be the same type.
- Must be the same total size (bytes).
- Must have identical dimensions (width, height, depth) or be a compatible [Reinterpret Copy](#).
- Must have compatible [DXGI formats](#), which means the formats must be identical or at least from the same type group. For example, a `DXGI_FORMAT_R32G32B32_FLOAT` texture can be copied to a `DXGI_FORMAT_R32G32B32_UINT` texture since both of these formats are in the `DXGI_FORMAT_R32G32B32_TYPELESS` group. **CopyResource** can copy between a few format types (see [Reinterpret copy](#)).
- Can't be currently mapped.

CopyResource only supports copy; it doesn't support any stretch, color key, or blend.

CopyResource can reinterpret the resource data between a few format types, see [Reinterpret Copy](#) below for details.

You can use a [depth-stencil](#) resource as either a source or a destination. Resources created with multi-sampling capability (see [DXGI_SAMPLE_DESC](#)) can be used as source and destination only if both source and destination have identical multi-sampled count and quality. If source and destination differ in multi-sampled count and quality or if one is multi-sampled and the other is not multi-sampled, the call to **CopyResource** fails. Use [ResolveSubresource](#) to resolve a multi-sampled resource to a resource that is not multi-sampled.

The method is an asynchronous call, which may be added to the command-buffer queue. This attempts to remove pipeline stalls that may occur when copying data. For more info, see [performance considerations](#).

Consider using [CopyTextureRegion](#) or [CopyBufferRegion](#) if you only need to copy a portion of the data in a resource.

Reinterpret copy

The following table lists the allowable source and destination formats that you can use in the reinterpretation type of format conversion. The underlying data values are not converted or compressed/decompressed and must be encoded properly for the reinterpretation to work as expected. For more info, see [Format Conversion using Direct3D 10.1](#).

For DXGI_FORMAT_R9G9B9E5_SHAREDEXP the width and height must be equal (1 texel per block).

Block-compressed resource width and height must be 4 times the uncompressed resource width and height (16 texels per block). For example, a uncompressed 256x256 DXGI_FORMAT_R32G32B32A32_UINT texture will map to a 1024x1024 DXGI_FORMAT_BC5_UNORM compressed texture.

[+] [Expand table](#)

Bit width	Uncompressed resource	Block-compressed resource	Width / height difference
32	DXGI_FORMAT_R32_UINT DXGI_FORMAT_R32_SINT	DXGI_FORMAT_R9G9B9E5_SHAREDEXP	1:1
64	DXGI_FORMAT_R16G16B16A16_UINT DXGI_FORMAT_R16G16B16A16_SINT DXGI_FORMAT_R32G32_UINT DXGI_FORMAT_R32G32_SINT	DXGI_FORMAT_BC1_UNORM[_SRGB] DXGI_FORMAT_BC4_UNORM DXGI_FORMAT_BC4_SNORM	1:4
128	DXGI_FORMAT_R32G32B32A32_UINT DXGI_FORMAT_R32G32B32A32_SINT	DXGI_FORMAT_BC2_UNORM[_SRGB] DXGI_FORMAT_BC3_UNORM[_SRGB] DXGI_FORMAT_BC5_UNORM DXGI_FORMAT_BC5_SNORM	1:4

Examples

The [D3D12HeterogeneousMultiadapter](#) sample uses [CopyResource](#) in the following way:

syntax

```

    // Command list to copy the render target to the shared heap on the
    // primary adapter.
    {
        const GraphicsAdapter adapter = Primary;

        // Reset the copy command allocator and command list.
        ThrowIfFailed(m_copyCommandAllocators[m_frameIndex]->Reset());
        ThrowIfFailed(m_copyCommandList-
>Reset(m_copyCommandAllocators[m_frameIndex].Get(), nullptr));

        // Copy the intermediate render target to the cross-adapter shared
        // resource.
        // Transition barriers are not required since there are fences
        // guarding against
        // concurrent read/write access to the shared heap.
        if (m_crossAdapterTextureSupport)
        {
            // If cross-adapter row-major textures are supported by the
            // adapter,
            // simply copy the texture into the cross-adapter texture.
            m_copyCommandList->CopyResource(m_crossAdapterResources[adapter]
[m_frameIndex].Get(), m_renderTargets[adapter][m_frameIndex].Get());
        }
        else
        {
            // If cross-adapter row-major textures are not supported by the
            // adapter,
            // the texture will be copied over as a buffer so that the
            // texture row
            // pitch can be explicitly managed.

            // Copy the intermediate render target into the shared buffer
            // using the
            // memory layout prescribed by the render target.
            D3D12_RESOURCE_DESC renderTargetDesc = m_renderTargets[adapter]
[m_frameIndex]->GetDesc();
            D3D12_PLACED_SUBRESOURCE_FOOTPRINT renderTargetLayout;

            m_devices[adapter]->GetCopyableFootprints(&renderTargetDesc, 0,
1, 0, &renderTargetLayout, nullptr, nullptr, nullptr);

            CD3DX12_TEXTURE_COPY_LOCATION
dest(m_crossAdapterResources[adapter][m_frameIndex].Get(),
renderTargetLayout);
            CD3DX12_TEXTURE_COPY_LOCATION src(m_renderTargets[adapter]
[m_frameIndex].Get(), 0);
            CD3DX12_BOX box(0, 0, m_width, m_height);

```

```
    m_copyCommandList->CopyTextureRegion(&dest, 0, 0, 0, &src,  
&box);  
}  
  
    ThrowIfFailed(m_copyCommandList->Close());  
}
```

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::CopyTextureRegion method (d3d12.h)

Article 10/13/2021

This method uses the GPU to copy texture data between two locations. Both the source and the destination may reference texture data located within either a buffer resource or a texture resource.

Syntax

C++

```
void CopyTextureRegion(
    [in]           const D3D12_TEXTURE_COPY_LOCATION *pDst,
                  UINT                         DstX,
                  UINT                         DstY,
                  UINT                         DstZ,
    [in]           const D3D12_TEXTURE_COPY_LOCATION *pSrc,
    [in, optional] const D3D12_BOX                 *pSrcBox
);
```

Parameters

[in] pDst

Type: [const D3D12_TEXTURE_COPY_LOCATION*](#)

Specifies the destination [D3D12_TEXTURE_COPY_LOCATION](#). The subresource referred to must be in the D3D12_RESOURCE_STATE_COPY_DEST state.

DstX

Type: [UINT](#)

The x-coordinate of the upper left corner of the destination region.

DstY

Type: [UINT](#)

The y-coordinate of the upper left corner of the destination region. For a 1D subresource, this must be zero.

DstZ

Type: **UINT**

The z-coordinate of the upper left corner of the destination region. For a 1D or 2D subresource, this must be zero.

[in] pSrc

Type: **const D3D12_TEXTURE_COPY_LOCATION***

Specifies the source **D3D12_TEXTURE_COPY_LOCATION**. The subresource referred to must be in the **D3D12_RESOURCE_STATE_COPY_SOURCE** state.

[in, optional] pSrcBox

Type: **const D3D12_BOX***

Specifies an optional **D3D12_BOX** that sets the size of the source texture to copy.

Return value

None

Remarks

The source box must be within the size of the source resource. The destination offsets, (x, y, and z), allow the source box to be offset when writing into the destination resource; however, the dimensions of the source box and the offsets must be within the size of the resource. If you try and copy outside the destination resource or specify a source box that is larger than the source resource, the behavior of **CopyTextureRegion** is undefined. If you created a device that supports the **debug layer**, the debug output reports an error on this invalid **CopyTextureRegion** call. Invalid parameters to **CopyTextureRegion** cause undefined behavior and might result in incorrect rendering, clipping, no copy, or even the removal of the rendering device.

If the resources are buffers, all coordinates are in bytes; if the resources are textures, all coordinates are in texels.

CopyTextureRegion performs the copy on the GPU (similar to a `memcpy` by the CPU). As a consequence, the source and destination resources:

- Must be different subresources (although they can be from the same resource).

- Must have compatible [DXGI_FORMAT](#)s (identical or from the same type group). For example, a DXGI_FORMAT_R32G32B32_FLOAT texture can be copied to a DXGI_FORMAT_R32G32B32_UINT texture since both of these formats are in the DXGI_FORMAT_R32G32B32_TYPELESS group. **CopyTextureRegion** can copy between a few format types. For more info, see [Format Conversion using Direct3D 10.1](#).

CopyTextureRegion only supports copy; it does not support any stretch, color key, or blend. **CopyTextureRegion** can reinterpret the resource data between a few format types.

Note that for a depth-stencil buffer, the depth and stencil planes are [separate subresources](#) within the buffer.

To copy an entire resource, rather than just a region of a subresource, we recommend to use [CopyResource](#) instead.

Note If you use **CopyTextureRegion** with a depth-stencil buffer or a multisampled resource, you must copy the entire subresource rectangle. In this situation, you must pass 0 to the *DstX*, *DstY*, and *DstZ* parameters and **NULL** to the *pSrcBox* parameter. In addition, source and destination resources, which are represented by the *pSrcResource* and *pDstResource* parameters, should have identical sample count values.

CopyTextureRegion may be used to initialize resources which alias the same heap memory. See [CreatePlacedResource](#) for more details.

Example

The following code snippet copies the box (located at (120,100),(200,220)) from a source texture into the region (10,20),(90,140) in a destination texture.

```
D3D12_BOX sourceRegion;
sourceRegion.left = 120;
sourceRegion.top = 100;
sourceRegion.right = 200;
sourceRegion.bottom = 220;
sourceRegion.front = 0;
sourceRegion.back = 1;

pCmdList -> CopyTextureRegion(pDestTexture, 10, 20, 0, pSourceTexture,
```

```
&sourceRegion);
```

Notice, that for a 2D texture, front and back are set to 0 and 1 respectively.

Examples

The HelloTriangle sample uses `ID3D12GraphicsCommandList::CopyTextureRegion` as follows:

C++

```
inline UINT64 UpdateSubresources(
    _In_ ID3D12GraphicsCommandList* pCmdList,
    _In_ ID3D12Resource* pDestinationResource,
    _In_ ID3D12Resource* pIntermediate,
    _In_range_(0,D3D12_REQ_SUBRESOURCES) UINT FirstSubresource,
    _In_range_(0,D3D12_REQ_SUBRESOURCES-FirstSubresource) UINT
NumSubresources,
    UINT64 RequiredSize,
    _In_reads_(NumSubresources) const D3D12_PLACED_SUBRESOURCE_FOOTPRINT*
pLayouts,
    _In_reads_(NumSubresources) const UINT* pNumRows,
    _In_reads_(NumSubresources) const UINT64* pRowSizesInBytes,
    _In_reads_(NumSubresources) const D3D12_SUBRESOURCE_DATA* pSrcData)
{
    // Minor validation
    D3D12_RESOURCE_DESC IntermediateDesc = pIntermediate->GetDesc();
    D3D12_RESOURCE_DESC DestinationDesc = pDestinationResource->GetDesc();
    if (IntermediateDesc.Dimension != D3D12_RESOURCE_DIMENSION_BUFFER ||
        IntermediateDesc.Width < RequiredSize + pLayouts[0].Offset ||
        RequiredSize > (SIZE_T)-1 ||
        (DestinationDesc.Dimension == D3D12_RESOURCE_DIMENSION_BUFFER &&
         (FirstSubresource != 0 || NumSubresources != 1)))
    {
        return 0;
    }

    BYTE* pData;
    HRESULT hr = pIntermediate->Map(0, NULL, reinterpret_cast<void**>
(&pData));
    if (FAILED(hr))
    {
        return 0;
    }

    for (UINT i = 0; i < NumSubresources; ++i)
    {
        if (pRowSizesInBytes[i] > (SIZE_T)-1) return 0;
        D3D12_MEMCPY_DEST DestData = { pData + pLayouts[i].Offset,
pLayouts[i].Footprint.RowPitch, pLayouts[i].Footprint.RowPitch * pNumRows[i]
```

```

};

        MemcpySubresource(&DestData, &pSrcData[i],
(SIZE_T)pRowSizesInBytes[i], pNumRows[i], pLayouts[i].Footprint.Depth);
    }

    pIntermediate->Unmap(0, NULL);

    if (DestinationDesc.Dimension == D3D12_RESOURCE_DIMENSION_BUFFER)
    {
        CD3DX12_BOX SrcBox( UINT( pLayouts[0].Offset ), UINT(  

pLayouts[0].Offset + pLayouts[0].Footprint.Width ) );
        pCmdList->CopyBufferRegion(
            pDestinationResource, 0, pIntermediate, pLayouts[0].Offset,
pLayouts[0].Footprint.Width);
    }
    else
    {
        for (UINT i = 0; i < NumSubresources; ++i)
        {
            CD3DX12_TEXTURE_COPY_LOCATION Dst(pDestinationResource, i +
FirstSubresource);
            CD3DX12_TEXTURE_COPY_LOCATION Src(pIntermediate, pLayouts[i]);
            pCmdList->CopyTextureRegion(&Dst, 0, 0, 0, &Src, nullptr);
        }
    }
    return RequiredSize;
}

```

See Example Code in the D3D12 Reference.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CopyBufferRegion](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::CopyTiles method (d3d12.h)

Article 07/27/2022

Copies tiles from buffer to tiled resource or vice versa.

Syntax

C++

```
void CopyTiles(  
    [in] ID3D12Resource                 *pTiledResource,  
    [in] const D3D12_TILED_RESOURCE_COORDINATE *pTileRegionStartCoordinate,  
    [in] const D3D12_TILE_REGION_SIZE        *pTileRegionSize,  
    [in] ID3D12Resource                 *pBuffer,  
    UINT64                            BufferStartOffsetInBytes,  
    D3D12_TILE_COPY_FLAGS                Flags  
) ;
```

Parameters

[in] pTiledResource

Type: **ID3D12Resource***

A pointer to a tiled resource.

[in] pTileRegionStartCoordinate

Type: **const D3D12_TILED_RESOURCE_COORDINATE***

A pointer to a **D3D12_TILED_RESOURCE_COORDINATE** structure that describes the starting coordinates of the tiled resource.

[in] pTileRegionSize

Type: **const D3D12_TILE_REGION_SIZE***

A pointer to a **D3D12_TILE_REGION_SIZE** structure that describes the size of the tiled region.

[in] pBuffer

Type: [ID3D12Resource*](#)

A pointer to an [ID3D12Resource](#) that represents a default, dynamic, or staging buffer.

`BufferStartOffsetInBytes`

Type: [UINT64](#)

The offset in bytes into the buffer at *pBuffer* to start the operation.

`Flags`

Type: [D3D12_TILE_COPY_FLAGS](#)

A combination of [D3D12_TILE_COPY_FLAGS](#)-typed values that are combined by using a bitwise OR operation and that identifies how to copy tiles.

Return value

None

Remarks

`CopyTiles` drops write operations to unmapped areas and handles read operations from unmapped areas (except on Tier_1 tiled resources, where reading and writing unmapped areas is invalid - refer to [D3D12_TILED_RESOURCES_TIER](#)).

If a copy operation involves writing to the same memory location multiple times because multiple locations in the destination resource are mapped to the same tile memory, the resulting write operations to multi-mapped tiles are non-deterministic and non-repeatable; that is, accesses to the tile memory happen in whatever order the hardware happens to execute the copy operation.

The tiles involved in the copy operation can't include tiles that contain packed mipmaps or results of the copy operation are undefined. To transfer data to and from mipmaps that the hardware packs into the one-or-more tiles that constitute the packed mips, you must use the standard (that is, non-tile specific) copy APIs like [CopyTextureRegion](#).

`CopyTiles` does copy data in a slightly different pattern than the standard copy methods.

The memory layout of the tiles in the non-tiled buffer resource side of the copy operation is linear in memory within 64 KB tiles, which the hardware and driver swizzle and de-swizzle per tile as appropriate when they transfer to and from a tiled resource.

For multisample antialiasing (MSAA) surfaces, the hardware and driver traverse each pixel's samples in sample-index order before they move to the next pixel. For tiles that are partially filled on the right side (for a surface that has a width not a multiple of tile width in pixels), the pitch and stride to move down a row is the full size in bytes of the number pixels that would fit across the tile if the tile was full. So, there can be a gap between each row of pixels in memory. Mipmaps that are smaller than a tile are not packed together in the linear layout, which might seem to be a waste of memory space, but as mentioned you can't use [CopyTiles](#) to copy to mipmaps that the hardware packs together. You can just use generic copy APIs, like [CopyTextureRegion](#), to copy small mipmaps individually.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Tiled resources](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::DiscardResource method (d3d12.h)

Article02/22/2024

Indicates that the contents of a resource don't need to be preserved. The function may re-initialize resource metadata in some cases.

Syntax

C++

```
void DiscardResource(  
    ID3D12Resource           *pResource,  
    const D3D12_DISCARD_REGION *pRegion  
) ;
```

Parameters

pResource

Type: [in] [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface for the resource to discard.

pRegion

Type: [in, optional] [const D3D12_DISCARD_REGION*](#)

A pointer to a [D3D12_DISCARD_REGION](#) structure that describes details for the discard-resource operation.

Return value

None

Remarks

The semantics of [DiscardResource](#) change based on the command list type.

For [D3D12_COMMAND_LIST_TYPE_DIRECT](#), the following two rules apply:

- When a resource has the [D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET](#) flag, [DiscardResource](#) must be called when the discarded subresource regions are in the [D3D12_RESOURCE_STATE_RENDER_TARGET](#) resource barrier state.
- When a resource has the [D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL](#) flag, [DiscardResource](#) must be called when the discarded subresource regions are in the [D3D12_RESOURCE_STATE_DEPTH_WRITE](#).

For [D3D12_COMMAND_LIST_TYPE_COMPUTE](#), the following rule applies:

- The resource must have the [D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS](#) flag, and [DiscardResource](#) must be called when the discarded subresource regions are in the [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#) resource barrier state.

[DiscardResource](#) is not supported on command lists with either [D3D12_COMMAND_LIST_TYPE_BUNDLE](#) nor [D3D12_COMMAND_LIST_TYPE_COPY](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::Dispatch method (d3d12.h)

Article 10/13/2021

Executes a command list from a thread group.

Syntax

C++

```
void Dispatch(
    [in] UINT ThreadGroupCountX,
    [in] UINT ThreadGroupCountY,
    [in] UINT ThreadGroupCountZ
);
```

Parameters

[in] ThreadGroupCountX

Type: **UINT**

The number of groups dispatched in the x direction. *ThreadGroupCountX* must be less than or equal to D3D11_CS_DISPATCH_MAX_THREAD_GROUPS_PER_DIMENSION (65535).

[in] ThreadGroupCountY

Type: **UINT**

The number of groups dispatched in the y direction. *ThreadGroupCountY* must be less than or equal to D3D11_CS_DISPATCH_MAX_THREAD_GROUPS_PER_DIMENSION (65535).

[in] ThreadGroupCountZ

Type: **UINT**

The number of groups dispatched in the z direction. *ThreadGroupCountZ* must be less than or equal to D3D11_CS_DISPATCH_MAX_THREAD_GROUPS_PER_DIMENSION (65535). In feature level 10 the value for *ThreadGroupCountZ* must be 1.

Return value

None

Remarks

You call the **Dispatch** method to execute commands in a compute shader. A compute shader can be run on many threads in parallel, within a thread group. Index a particular thread, within a thread group using a 3D vector given by (x,y,z).

Examples

The [D3D12nBodyGravity](#) sample uses **ID3D12GraphicsCommandList::Dispatch** as follows:

C++

```
// Run the particle simulation using the compute shader.
void D3D12nBodyGravity::Simulate(UINT threadIndex)
{
    ID3D12GraphicsCommandList* pCommandList =
    m_computeCommandList[threadIndex].Get();

    UINT srvIndex;
    UINT uavIndex;
    ID3D12Resource *pUavResource;
    if (m_srvIndex[threadIndex] == 0)
    {
        srvIndex = SrvParticlePosVelo0;
        uavIndex = UavParticlePosVelo1;
        pUavResource = m_particleBuffer1[threadIndex].Get();
    }
    else
    {
        srvIndex = SrvParticlePosVelo1;
        uavIndex = UavParticlePosVelo0;
        pUavResource = m_particleBuffer0[threadIndex].Get();
    }

    pCommandList->ResourceBarrier(1,
    &CD3DX12_RESOURCE_BARRIER::Transition(pUavResource,
    D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
    D3D12_RESOURCE_STATE_UNORDERED_ACCESS));

    pCommandList->SetPipelineState(m_computeState.Get());
    pCommandList->SetComputeRootSignature(m_computeRootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_srvUavHeap.Get() };
    pCommandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);
```

```

    CD3DX12_GPU_DESCRIPTOR_HANDLE srvHandle(m_srvUavHeap-
>GetGPUDescriptorHandleForHeapStart(), srvIndex + threadIndex,
m_srvUavDescriptorSize);
    CD3DX12_GPU_DESCRIPTOR_HANDLE uavHandle(m_srvUavHeap-
>GetGPUDescriptorHandleForHeapStart(), uavIndex + threadIndex,
m_srvUavDescriptorSize);

        pCommandList->SetComputeRootConstantBufferView(RootParameterCB,
m_constantBufferCS->GetGPUVirtualAddress());
        pCommandList->SetComputeRootDescriptorTable(RootParameterSRV,
srvHandle);
        pCommandList->SetComputeRootDescriptorTable(RootParameterUAV,
uavHandle);

    pCommandList->Dispatch(static_cast<int>(ceil(ParticleCount / 128.0f)),
1, 1);

    pCommandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(pUavResource,
D3D12_RESOURCE_STATE_UNORDERED_ACCESS,
D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE));
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::DrawIndexedInstanced method (d3d12.h)

Article 02/22/2024

Draws indexed, instanced primitives.

Syntax

C++

```
void DrawIndexedInstanced(
    [in] UINT IndexCountPerInstance,
    [in] UINT InstanceCount,
    [in] UINT StartIndexLocation,
    [in] INT BaseVertexLocation,
    [in] UINT StartInstanceLocation
);
```

Parameters

[in] IndexCountPerInstance

Type: **UINT**

Number of indices read from the index buffer for each instance.

[in] InstanceCount

Type: **UINT**

Number of instances to draw.

[in] StartIndexLocation

Type: **UINT**

The location of the first index read by the GPU from the index buffer.

[in] BaseVertexLocation

Type: **INT**

A value added to each index before reading a vertex from the vertex buffer.

[in] StartInstanceLocation

Type: **UINT**

A value added to each index before reading per-instance data from a vertex buffer.

Return value

None

Remarks

A draw API submits work to the rendering pipeline.

Instancing might extend performance by reusing the same geometry to draw multiple objects in a scene. One example of instancing could be to draw the same object with different positions and colors. Instancing requires multiple vertex buffers: at least one for per-vertex data and a second buffer for per-instance data.

Examples

The [D3D12Bundles](#) sample uses `ID3D12GraphicsCommandList::DrawIndexedInstanced` as follows:

C++

```
void FrameResource::PopulateCommandList(ID3D12GraphicsCommandList*
pCommandList, ID3D12PipelineState* pPso1, ID3D12PipelineState* pPso2,
    UINT frameResourceIndex, UINT numIndices, D3D12_INDEX_BUFFER_VIEW*
pIndexBufferViewDesc, D3D12_VERTEX_BUFFER_VIEW* pVertexBufferViewDesc,
    ID3D12DescriptorHeap* pCbvSrvDescriptorHeap, UINT cbvSrvDescriptorSize,
ID3D12DescriptorHeap* pSamplerDescriptorHeap, ID3D12RootSignature*
pRootSignature)
{
    // If the root signature matches the root signature of the caller, then
    // bindings are inherited, otherwise the bind space is reset.
    pCommandList->SetGraphicsRootSignature(pRootSignature);

    ID3D12DescriptorHeap* ppHeaps[] = { pCbvSrvDescriptorHeap,
    pSamplerDescriptorHeap };
    pCommandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);
    pCommandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

    pCommandList->IASetIndexBuffer(pIndexBufferViewDesc);

    pCommandList->IASetVertexBuffers(0, 1, pVertexBufferViewDesc);
```

```

    pCommandList->SetGraphicsRootDescriptorTable(0, pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());
    pCommandList->SetGraphicsRootDescriptorTable(1, pSamplerDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());

    // Calculate the descriptor offset due to multiple frame resources.
    // 1 SRV + how many CBVs we have currently.
    UINT frameResourceDescriptorOffset = 1 + (frameResourceIndex *
m_cityRowCount * m_cityColumnCount);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvSrvHandle(pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart(), frameResourceDescriptorOffset,
cbvSrvDescriptorSize);

    BOOL usePso1 = TRUE;
    for (UINT i = 0; i < m_cityRowCount; i++)
    {
        for (UINT j = 0; j < m_cityColumnCount; j++)
        {
            // Alternate which PSO to use; the pixel shader is different on
            // each just as a PSO setting demonstration.
            pCommandList->SetPipelineState(usePso1 ? pPso1 : pPso2);
            usePso1 = !usePso1;

            // Set this city's CBV table and move to the next descriptor.
            pCommandList->SetGraphicsRootDescriptorTable(2, cbvSrvHandle);
            cbvSrvHandle.Offset(cbvSrvDescriptorSize);

            pCommandList->DrawIndexedInstanced(numIndices, 1, 0, 0, 0);
        }
    }
}

```

See Example Code in the D3D12 Reference.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::DrawInstanced method (d3d12.h)

Article 02/22/2024

Draws non-indexed, instanced primitives.

Syntax

C++

```
void DrawInstanced(
    [in] UINT VertexCountPerInstance,
    [in] UINT InstanceCount,
    [in] UINT StartVertexLocation,
    [in] UINT StartInstanceLocation
);
```

Parameters

[in] VertexCountPerInstance

Type: [UINT](#)

Number of vertices to draw.

[in] InstanceCount

Type: [UINT](#)

Number of instances to draw.

[in] StartVertexLocation

Type: [UINT](#)

Index of the first vertex.

[in] StartInstanceLocation

Type: [UINT](#)

A value added to each index before reading per-instance data from a vertex buffer.

Return value

None

Remarks

A draw API submits work to the rendering pipeline.

Instancing might extend performance by reusing the same geometry to draw multiple objects in a scene. One example of instancing could be to draw the same object with different positions and colors.

The vertex data for an instanced draw call typically comes from a vertex buffer that is bound to the pipeline. But, you could also provide the vertex data from a shader that has instanced data identified with a system-value semantic (SV_InstanceID).

Examples

The [D3D12HelloTriangle](#) sample uses `ID3D12GraphicsCommandList::DrawInstanced` as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());
```

```

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
before
    // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
    m_commandList->DrawInstanced(3, 1, 0, 0);

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

Requirement	Value
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::EndEvent method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command list.

Syntax

C++

```
void EndEvent();
```

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To mark the end of an instrumentation region at the current location within a D3D12 command list, use the **PIXEndEvent** function or **PIXScopedEvent** macro. These are provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::EndQuery method (d3d12.h)

Article 02/22/2024

Ends a running query.

Syntax

C++

```
void EndQuery(
    [in] ID3D12QueryHeap  *pQueryHeap,
    [in] D3D12_QUERY_TYPE Type,
    [in] UINT              Index
);
```

Parameters

[in] pQueryHeap

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the query.

[in] Type

Type: [D3D12_QUERY_TYPE](#)

Specifies one member of [D3D12_QUERY_TYPE](#).

[in] Index

Type: [UINT](#)

Specifies the index of the query in the query heap.

Return value

None

Remarks

See [Queries](#) for more information about D3D12 queries.

Examples

The [D3D12PredicationQueries](#) sample uses `ID3D12GraphicsCommandList::EndQuery` as follows:

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12PredicationQueries::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
before
    // re-recording.
    ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvHeap.Get() };
    m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList->ClearDepthStencilView(dsvHandle, D3D12_CLEAR_FLAG_DEPTH,
1.0f, 0, 0, nullptr);
```

```

// Draw the quads and perform the occlusion query.
{
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvFarQuad(m_cbvHeap-
>GetGPUDescriptorHandleForHeapStart(), m_frameIndex * CbvCountPerFrame,
m_cbvSrvDescriptorSize);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvNearQuad(cbvFarQuad,
m_cbvSrvDescriptorSize);

    m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);

        // Draw the far quad conditionally based on the result of the
occlusion query
        // from the previous frame.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPredication(m_queryResult.Get(), 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->DrawInstanced(4, 1, 0, 0);

        // Disable predication and always draw the near quad.
        m_commandList->SetPredication(nullptr, 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvNearQuad);
        m_commandList->DrawInstanced(4, 1, 4, 0);

        // Run the occlusion query with the bounding box quad.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPipelineState(m_queryState.Get());
        m_commandList->BeginQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);
        m_commandList->DrawInstanced(4, 1, 8, 0);
        m_commandList->EndQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);

        // Resolve the occlusion query and store the results in the query
result buffer
        // to be used on the subsequent frame.
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_PREDICATION, D3D12_RESOURCE_STATE_COPY_DEST));
        m_commandList->ResolveQueryData(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0, 1, m_queryResult.Get(), 0);
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_PREDICATION));
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

}

See Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ExecuteBundle method (d3d12.h)

Article 10/13/2021

Executes a bundle.

Syntax

C++

```
void ExecuteBundle(  
    [in] ID3D12GraphicsCommandList *pCommandList  
);
```

Parameters

[in] pCommandList

Type: [ID3D12GraphicsCommandList*](#)

Specifies the [ID3D12GraphicsCommandList](#) that determines the bundle to be executed.

Return value

None

Remarks

Bundles inherit all state from the parent command list on which **ExecuteBundle** is called, except the pipeline state object and primitive topology. All of the state that is set in a bundle will affect the state of the parent command list. Note that **ExecuteBundle** is not a predicated operation.

Runtime validation

The runtime will validate that the "callee" is a bundle and that the "caller" is a direct command list. The runtime will also validate that the bundle has been closed. If the

contract is violated, the runtime will silently drop the call. Validation failure will result in [Close](#) returning E_INVALIDARG.

Debug layer

The debug layer will issue a warning in the same cases where the runtime will fail. The debug layer will issue a warning if a predicate is set when [ExecuteCommandList](#) is called. Also, the debug layer will issue an error if it detects that any resource reference by the command list has been destroyed.

The debug layer will also validate that the command allocator associated with the bundle has not been reset since [Close](#) was called on the command list. This validation occurs at [ExecuteBundle](#) time, and when the parent command list is executed on a command queue.

Examples

The [D3D12Bundles](#) sample uses [ID3D12GraphicsCommandList::ExecuteBundle](#) as follows:

C++

```
void D3D12Bundles::PopulateCommandList(FrameResource* pFrameResource)
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_pCurrentFrameResource->m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
    // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_pCurrentFrameResource-
>m_commandAllocator.Get(), m_pipelineState1.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvSrvHeap.Get(),
m_samplerHeap.Get() };
    m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
```

```

&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList->ClearDepthStencilView(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart(), D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0,
nullptr);

    if (UseBundles)
    {
        // Execute the prebuilt bundle.
        m_commandList->ExecuteBundle(pFrameResource->m_bundle.Get());
    }
    else
    {
        // Populate a new command list.
        pFrameResource->PopulateCommandList(m_commandList.Get(),
m_pipelineState1.Get(), m_pipelineState2.Get(), m_currentFrameResourceIndex,
m_numIndices, &m_indexBufferView,
&m_vertexBufferView, m_cbvSrvHeap.Get(), m_cbvSrvDescriptorSize,
m_samplerHeap.Get(), m_rootSignature.Get());
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ExecuteIndirect method (d3d12.h)

Article 10/13/2021

Apps perform indirect draws/dispatches using the `ExecuteIndirect` method.

Syntax

C++

```
void ExecuteIndirect(
    [in]          ID3D12CommandSignature *pCommandSignature,
    [in]          UINT                  MaxCommandCount,
    [in]          ID3D12Resource       *pArgumentBuffer,
    [in]          UINT64               ArgumentBufferOffset,
    [in, optional] ID3D12Resource     *pCountBuffer,
    [in]          UINT64               CountBufferOffset
);
```

Parameters

[in] `pCommandSignature`

Type: [ID3D12CommandSignature*](#)

Specifies a [ID3D12CommandSignature](#). The data referenced by *pArgumentBuffer* will be interpreted depending on the contents of the command signature. Refer to [Indirect Drawing](#) for the APIs that are used to create a command signature.

[in] `MaxCommandCount`

Type: `UINT`

There are two ways that command counts can be specified:

- If *pCountBuffer* is not NULL, then *MaxCommandCount* specifies the maximum number of operations which will be performed. The actual number of operations to be performed are defined by the minimum of this value, and a 32-bit unsigned integer contained in *pCountBuffer* (at the byte offset specified by *CountBufferOffset*).
- If *pCountBuffer* is NULL, the *MaxCommandCount* specifies the exact number of operations which will be performed.

[in] *pArgumentBuffer*

Type: [ID3D12Resource*](#)

Specifies one or more [ID3D12Resource](#) objects, containing the command arguments.

[in] *ArgumentBufferOffset*

Type: [UINT64](#)

Specifies an offset into *pArgumentBuffer* to identify the first command argument.

[in, optional] *pCountBuffer*

Type: [ID3D12Resource*](#)

Specifies a pointer to a [ID3D12Resource](#).

[in] *CountBufferOffset*

Type: [UINT64](#)

Specifies a [UINT64](#) that is the offset into *pCountBuffer*, identifying the argument count.

Return value

None

Remarks

The semantics of this API are defined with the following pseudo-code:

Non-NULL *pCountBuffer*:

syntax

```
// Read draw count out of count buffer
UINT CommandCount = pCountBuffer->ReadUINT32(CountBufferOffset);

CommandCount = min(CommandCount, MaxCommandCount)

// Get pointer to first Commanding argument
BYTE* Arguments = pArgumentBuffer->GetBase() + ArgumentBufferOffset;

for(UINT CommandIndex = 0; CommandIndex < CommandCount; CommandIndex++)
{
    // Interpret the data contained in *Arguments
    // according to the command signature
```

```
pCommandSignature->Interpret(Arguments);

    Arguments += pCommandSignature->GetByteStride();
}
```

NULL *pCountBuffer*:

syntax

```
// Get pointer to first Commanding argument
BYTE* Arguments = pArgumentBuffer->GetBase() + ArgumentBufferOffset;

for(UINT CommandIndex = 0; CommandIndex < MaxCommandCount; CommandIndex++)
{
    // Interpret the data contained in *Arguments
    // according to the command signature
    pCommandSignature->Interpret(Arguments);

    Arguments += pCommandSignature->GetByteStride();
}
```

The debug layer will issue an error if either the count buffer or the argument buffer are not in the D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT state. The core runtime will validate:

- *CountBufferOffset* and *ArgumentBufferOffset* are 4-byte aligned
- *pCountBuffer* and *pArgumentBuffer* are buffer resources (any heap type)
- The offset implied by *MaxCommandCount*, *ArgumentBufferOffset*, and the drawing program stride do not exceed the bounds of *pArgumentBuffer* (similarly for count buffer)
- The command list is a direct command list or a compute command list (not a copy or JPEG decode command list)
- The root signature of the command list matches the root signature of the command signature

The functionality of two APIs from earlier versions of Direct3D, `DrawInstancedIndirect` and `DrawIndexedInstancedIndirect`, are encompassed by `ExecuteIndirect`.

Bundles

`ID3D12GraphicsCommandList::ExecuteIndirect` is allowed inside of bundle command lists only if all of the following are true:

- CountBuffer is NULL (CPU-specified count only).

- The command signature contains exactly one operation. This implies that the command signature does not contain root arguments changes, nor contain VB/IB binding changes.

Obtaining buffer virtual addresses

The [ID3D12Resource::GetGPUVirtualAddress](#) method enables an app to retrieve the GPU virtual address of a buffer.

Apps are free to apply byte offsets to virtual addresses before placing them in an indirect argument buffer. Note that all of the D3D12 alignment requirements for VB/IB/CB still apply to the resulting GPU virtual address.

Examples

The [D3D12ExecuteIndirect](#) sample uses [ID3D12GraphicsCommandList::ExecuteIndirect](#) as follows:

C++

```
// Data structure to match the command signature used for ExecuteIndirect.
struct IndirectCommand
{
    D3D12_GPU_VIRTUAL_ADDRESS cbv;
    D3D12_DRAW_ARGUMENTS drawArguments;
};
```

The call to **ExecuteIndirect** is near the end of this listing, below the comment "Draw the triangles that have not been culled."

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12ExecuteIndirect::PopulateCommandLists()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_computeCommandAllocators[m_frameIndex]->Reset());
    ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    // before
    // re-recording.
    ThrowIfFailed(m_computeCommandList-
```

```

>Reset(m_computeCommandAllocators[m_frameIndex].Get(),
m_computeState.Get());
    ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

    // Record the compute commands that will cull triangles and prevent them
from being processed by the vertex shader.
    if (m_enableCulling)
    {
        UINT frameDescriptorOffset = m_frameIndex *
CbvSrvUavDescriptorCountPerFrame;
        D3D12_GPU_DESCRIPTOR_HANDLE cbvSrvUavHandle = m_cbvSrvUavHeap-
>GetGPUDescriptorHandleForHeapStart();

        m_computeCommandList-
>SetComputeRootSignature(m_computeRootSignature.Get());

        ID3D12DescriptorHeap* ppHeaps[] = { m_cbvSrvUavHeap.Get() };
        m_computeCommandList->SetDescriptorHeaps(_countof(ppHeaps),
ppHeaps);

        m_computeCommandList->SetComputeRootDescriptorTable(
            SrvUavTable,
            CD3DX12_GPU_DESCRIPTOR_HANDLE(cbvSrvUavHandle, CbvSrvOffset +
frameDescriptorOffset, m_cbvSrvUavDescriptorSize));

        m_computeCommandList->SetComputeRoot32BitConstants(RootConstants, 4,
reinterpret_cast<void*>(&m_csRootConstants), 0);

        // Reset the UAV counter for this frame.
        m_computeCommandList-
>CopyBufferRegion(m_processedCommandBuffers[m_frameIndex].Get(),
CommandBufferSizePerFrame, m_processedCommandBufferCounterReset.Get(), 0,
sizeof(UINT));

        D3D12_RESOURCE_BARRIER barrier =
CD3DX12_RESOURCE_BARRIER::Transition(m_processedCommandBuffers[m_frameIndex]
.Get(), D3D12_RESOURCE_STATE_COPY_DEST,
D3D12_RESOURCE_STATE_UNORDERED_ACCESS);
        m_computeCommandList->ResourceBarrier(1, &barrier);

        m_computeCommandList->Dispatch(static_cast<UINT>(ceil(TriangleCount
/ float(ComputeThreadBlockSize))), 1, 1);
    }

    ThrowIfFailed(m_computeCommandList->Close());

    // Record the rendering commands.
    {
        // Set necessary state.
        m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

        ID3D12DescriptorHeap* ppHeaps[] = { m_cbvSrvUavHeap.Get() };
        m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

```

```

        m_commandList->RSSetViewports(1, &m_viewport);
        m_commandList->RSSetScissorRects(1, m_enableCulling ?
&m_cullingScissorRect : &m_scissorRect);

        // Indicate that the command buffer will be used for indirect
drawing
        // and that the back buffer will be used as a render target.
D3D12_RESOURCE_BARRIER barriers[2] = {
    CD3DX12_RESOURCE_BARRIER::Transition(
        m_enableCulling ?

m_processedCommandBuffers[m_frameIndex].Get() : m_commandBuffer.Get(),
        m_enableCulling ? D3D12_RESOURCE_STATE_UNORDERED_ACCESS :
D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,
        D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT),
    CD3DX12_RESOURCE_BARRIER::Transition(
        m_renderTargets[m_frameIndex].Get(),
        D3D12_RESOURCE_STATE_PRESENT,
        D3D12_RESOURCE_STATE_RENDER_TARGET)
};

m_commandList->ResourceBarrier(_countof(barriers), barriers);

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0,
nullptr);
m_commandList->ClearDepthStencilView(dsvHandle,
D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0, nullptr);

m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);

if (m_enableCulling)
{
    // Draw the triangles that have not been culled.
    m_commandList->ExecuteIndirect(
        m_commandSignature.Get(),
        TriangleCount,
        m_processedCommandBuffers[m_frameIndex].Get(),
        0,
        m_processedCommandBuffers[m_frameIndex].Get(),
        CommandBufferSizePerFrame);
}
else
{
    // Draw all of the triangles.
    m_commandList->ExecuteIndirect(
        m_commandSignature.Get(),

```

```

        TriangleCount,
        m_commandBuffer.Get(),
        CommandBufferSizePerFrame * m_frameIndex,
        nullptr,
        0);
    }

    // Indicate that the command buffer may be used by the compute
    shader
    // and that the back buffer will now be used to present.
    barriers[0].Transition.StateBefore =
D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT;
    barriers[0].Transition.StateAfter = m_enableCulling ?
D3D12_RESOURCE_STATE_COPY_DEST :
D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE;
    barriers[1].Transition.StateBefore =
D3D12_RESOURCE_STATE_RENDER_TARGET;
    barriers[1].Transition.StateAfter = D3D12_RESOURCE_STATE_PRESENT;

    m_commandList->ResourceBarrier(_countof(barriers), barriers);

    ThrowIfFailed(m_commandList->Close());
}
}

```

See Example Code in the [D3D12 Reference](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Indirect Drawing](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::IASetIndexBuffer method (d3d12.h)

Article 02/22/2024

Sets the view for the index buffer.

Syntax

C++

```
void IASetIndexBuffer(  
    [in, optional] const D3D12_INDEX_BUFFER_VIEW *pView  
);
```

Parameters

[in, optional] pView

Type: [const D3D12_INDEX_BUFFER_VIEW*](#)

The view specifies the index buffer's address, size, and [DXGI_FORMAT](#), as a pointer to a [D3D12_INDEX_BUFFER_VIEW](#) structure.

Return value

None

Remarks

Only one index buffer can be bound to the graphics pipeline at any one time.

Examples

The [D3D12Bundles](#) sample uses [ID3D12GraphicsCommandList::IASetIndexBuffer](#) as follows:

C++

```

void FrameResource::PopulateCommandList(ID3D12GraphicsCommandList*
pCommandList, ID3D12PipelineState* pPso1, ID3D12PipelineState* pPso2,
    UINT frameResourceIndex, UINT numIndices, D3D12_INDEX_BUFFER_VIEW*
pIndexBufferViewDesc, D3D12_VERTEX_BUFFER_VIEW* pVertexBufferViewDesc,
    ID3D12DescriptorHeap* pCbvSrvDescriptorHeap, UINT cbvSrvDescriptorSize,
ID3D12DescriptorHeap* pSamplerDescriptorHeap, ID3D12RootSignature*
pRootSignature)
{
    // If the root signature matches the root signature of the caller, then
    // bindings are inherited, otherwise the bind space is reset.
    pCommandList->SetGraphicsRootSignature(pRootSignature);

    ID3D12DescriptorHeap* ppHeaps[] = { pCbvSrvDescriptorHeap,
pSamplerDescriptorHeap };
    pCommandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);
    pCommandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

    pCommandList->IASetIndexBuffer(pIndexBufferViewDesc);

    pCommandList->IASetVertexBuffers(0, 1, pVertexBufferViewDesc);

    pCommandList->SetGraphicsRootDescriptorTable(0, pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());
    pCommandList->SetGraphicsRootDescriptorTable(1, pSamplerDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());

    // Calculate the descriptor offset due to multiple frame resources.
    // 1 SRV + how many CBVs we have currently.
    UINT frameResourceDescriptorOffset = 1 + (frameResourceIndex *
m_cityRowCount * m_cityColumnCount);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvSrvHandle(pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart(), frameResourceDescriptorOffset,
cbvSrvDescriptorSize);

    BOOL usePso1 = TRUE;
    for (UINT i = 0; i < m_cityRowCount; i++)
    {
        for (UINT j = 0; j < m_cityColumnCount; j++)
        {
            // Alternate which PSO to use; the pixel shader is different on
            // each just as a PSO setting demonstration.
            pCommandList->SetPipelineState(usePso1 ? pPso1 : pPso2);
            usePso1 = !usePso1;

            // Set this city's CBV table and move to the next descriptor.
            pCommandList->SetGraphicsRootDescriptorTable(2, cbvSrvHandle);
            cbvSrvHandle.Offset(cbvSrvDescriptorSize);

            pCommandList->DrawIndexedInstanced(numIndices, 1, 0, 0, 0);
        }
    }
}

```

}

See Example Code in the D3D12 Reference.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::IASetPrimitiveTopology method (d3d12.h)

Article 02/22/2024

Bind information about the primitive type, and data order that describes input data for the input assembler stage.

Syntax

C++

```
void IASetPrimitiveTopology(  
    [in] D3D12_PRIMITIVE_TOPOLOGY PrimitiveTopology  
);
```

Parameters

[in] PrimitiveTopology

Type: [D3D12_PRIMITIVE_TOPOLOGY](#)

The type of primitive and ordering of the primitive data (see [D3D_PRIMITIVE_TOPOLOGY](#)).

Return value

None

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

Requirement	Value
DLL	D3d12.dll

See also

[D3D12_PRIMITIVE_TOPOLOGY_TYPE](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::IASetVertexBuffers method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the vertex buffers.

Syntax

C++

```
void IASetVertexBuffers(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [in, optional] const D3D12_VERTEX_BUFFER_VIEW *pViews
);
```

Parameters

[in] StartSlot

Type: **UINT**

Index into the device's zero-based array to begin setting vertex buffers.

[in] NumViews

Type: **UINT**

The number of views in the *pViews* array.

[in, optional] pViews

Type: **const D3D12_VERTEX_BUFFER_VIEW***

Specifies the vertex buffer views in an array of **D3D12_VERTEX_BUFFER_VIEW** structures.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[IASetIndexBuffer](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::OMSetBlendFactor method (d3d12.h)

Article 02/22/2024

Sets the blend factor that modulate values for a pixel shader, render target, or both.

Syntax

C++

```
void OMSetBlendFactor(  
    [in, optional] const FLOAT [4] BlendFactor  
);
```

Parameters

[in, optional] BlendFactor

Type: **const FLOAT[4]**

Array of blend factors, one for each RGBA component.

Return value

None

Remarks

If you created the blend-state object with [D3D12_BLEND_BLEND_FACTOR](#) or [D3D12_BLEND_INV_BLEND_FACTOR](#), then the blending stage uses the non-NULL array of blend factors. Otherwise, the blending stage doesn't use the non-NULL array of blend factors; the runtime stores the blend factors.

If you pass NULL, then the runtime uses or stores a blend factor equal to { 1, 1, 1, 1 }.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::OMSetRenderTargetTargets method (d3d12.h)

Article 10/13/2021

Sets CPU descriptor handles for the render targets and depth stencil.

Syntax

C++

```
void OMSetRenderTargetTargets(
    [in]          UINT
    NumRenderTargetDescriptors,
    [in, optional] const D3D12_CPU_DESCRIPTOR_HANDLE
    *pRenderTargetDescriptors,
    [in]          BOOL
    RTsSingleHandleToDescriptorRange,
    [in, optional] const D3D12_CPU_DESCRIPTOR_HANDLE *pDepthStencilDescriptor
);
```

Parameters

[in] NumRenderTargetDescriptors

Type: **UINT**

The number of entries in the *pRenderTargetDescriptors* array (ranges between 0 and **D3D12_SIMULTANEOUS_RENDER_TARGET_COUNT**). If this parameter is nonzero, the number of entries in the array to which *pRenderTargetDescriptors* points must equal the number in this parameter.

[in, optional] pRenderTargetDescriptors

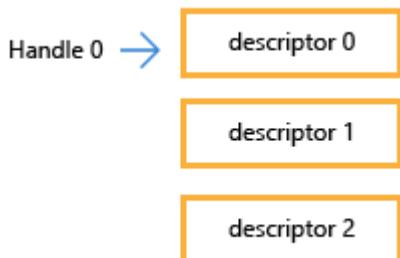
Type: **const D3D12_CPU_DESCRIPTOR_HANDLE***

Specifies an array of **D3D12_CPU_DESCRIPTOR_HANDLE** structures that describe the CPU descriptor handles that represents the start of the heap of render target descriptors. If this parameter is NULL and *NumRenderTargetDescriptors* is 0, no render targets are bound.

[in] RTsSingleHandleToDescriptorRange

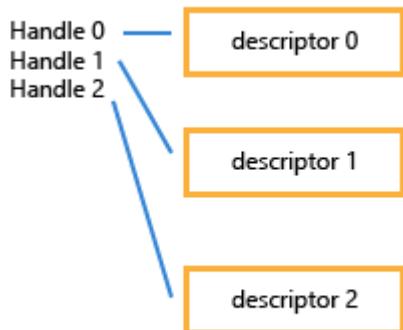
Type: **BOOL**

True means the handle passed in is the pointer to a contiguous range of *NumRenderTargetDescriptors* descriptors. This case is useful if the set of descriptors to bind already happens to be contiguous in memory (so all that's needed is a handle to the first one). For example, if *NumRenderTargetDescriptors* is 3 then the memory layout is taken as follows:



In this case the driver dereferences the handle and then increments the memory being pointed to.

False means that the handle is the first of an array of *NumRenderTargetDescriptors* handles. The false case allows an application to bind a set of descriptors from different locations at once. Again assuming that *NumRenderTargetDescriptors* is 3, the memory layout is taken as follows:



In this case the driver dereferences three handles that are expected to be adjacent to each other in memory.

`[in, optional] pDepthStencilDescriptor`

Type: **const D3D12_CPU_DESCRIPTOR_HANDLE***

A pointer to a [D3D12_CPU_DESCRIPTOR_HANDLE](#) structure that describes the CPU descriptor handle that represents the start of the heap that holds the depth stencil descriptor. If this parameter is NULL, no depth stencil descriptor is bound.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::OMSetStencilRef method (d3d12.h)

Article 02/22/2024

Sets the reference value for depth stencil tests.

Syntax

C++

```
void OMSetStencilRef(  
    [in] UINT StencilRef  
);
```

Parameters

[in] StencilRef

Type: **UINT**

Reference value to perform against when doing a depth-stencil test.

Return value

None

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::Reset method (d3d12.h)

Article 06/26/2024

Resets a command list back to its initial state as if a new command list was just created.

Syntax

C++

```
HRESULT Reset(
    [in]          ID3D12CommandAllocator *pAllocator,
    [in, optional] ID3D12PipelineState   *pInitialState
);
```

Parameters

[in] pAllocator

Type: [ID3D12CommandAllocator*](#)

A pointer to the [ID3D12CommandAllocator](#) object that the device creates command lists from.

[in, optional] pInitialState

Type: [ID3D12PipelineState*](#)

A pointer to the [ID3D12PipelineState](#) object that contains the initial pipeline state for the command list. This is optional and can be NULL. If NULL, the runtime sets a dummy initial pipeline state so that drivers don't have to deal with undefined state. The overhead for this is low, particularly for a command list, for which the overall cost of recording the command list likely dwarfs the cost of one initial state setting. So there is little cost in not setting the initial pipeline state parameter if it isn't convenient.

For bundles on the other hand, it might make more sense to try to set the initial state parameter since bundles are likely smaller overall and can be reused frequently.

Return value

Type: [HRESULT](#)

Returns **S_OK** if successful; otherwise, returns one of the following values:

- **E_FAIL** if the command list was not in the "closed" state when the **Reset** call was made, or the per-device limit would have been exceeded.
- **E_OUTOFMEMORY** if the operating system ran out of memory.
- **E_INVALIDARG** if the allocator is currently being used with another command list in the "recording" state or if the specified allocator was created with the wrong type.

See [Direct3D 12 Return Codes](#) for other possible return values.

Remarks

By using **Reset**, you can re-use command list tracking structures without any allocations. Unlike [ID3D12CommandAllocator::Reset](#), you can call **Reset** while the command list is still being executed.

You can use **Reset** for both direct command lists and bundles.

The command allocator passed to **Reset** cannot be associated with any other currently-recording command list. The allocator type, direct command list or bundle, must match the type of command list that is being created.

If a bundle doesn't specify a resource heap, it can't make changes to which descriptor tables are bound. Either way, bundles can't change the resource heap within the bundle. If a heap is specified for a bundle, the heap must match the calling 'parent' command list's heap.

Runtime validation

Before an app calls **Reset**, the command list must be in the "closed" state. **Reset** will fail if the command list isn't in the "closed" state.

Note If a call to [ID3D12GraphicsCommandList::Close](#) fails, the command list can never be reset. Calling **Reset** will result in the same error being returned that [ID3D12GraphicsCommandList::Close](#) returned.

After **Reset** succeeds, the command list is left in the "recording" state. **Reset** will fail if it would cause the maximum concurrently recording command list limit, which is specified

at device creation, to be exceeded.

Apps must specify a command list allocator. The runtime will ensure that an allocator is never associated with more than one recording command list at the same time.

Reset fails for bundles that are referenced by a not yet submitted command list.

Debug layer

The debug layer will also track graphics processing unit (GPU) progress and issue an error if it can't prove that there are no outstanding executions of the command list.

Examples

The [D3D12HelloTriangle](#) sample uses **ID3D12GraphicsCommandList::Reset** as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
        // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
    m_pipelineState.Get()));

    // Set necessary state.
```

```

m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
m_commandList->RSSetViewports(1, &m_viewport);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

ThrowIfFailed(m_commandList->Close());
}

```

See Example Code in the D3D12 Reference.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandAllocator::Reset](#)

[ID3D12Device::CreateCommandList](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ResolveQueryData method (d3d12.h)

Article 10/13/2021

Extracts data from a query. `ResolveQueryData` works with all heap types (default, upload, and readback).

Syntax

C++

```
void ResolveQueryData(
    [in] ID3D12QueryHeap  *pQueryHeap,
    [in] D3D12_QUERY_TYPE Type,
    [in] UINT             StartIndex,
    [in] UINT             NumQueries,
    [in] ID3D12Resource   *pDestinationBuffer,
    [in] UINT64           AlignedDestinationBufferOffset
);
```

Parameters

[in] `pQueryHeap`

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the queries to resolve.

[in] `Type`

Type: [D3D12_QUERY_TYPE](#)

Specifies the type of query, one member of [D3D12_QUERY_TYPE](#).

[in] `StartIndex`

Type: [UINT](#)

Specifies an index of the first query to resolve.

[in] `NumQueries`

Type: [UINT](#)

Specifies the number of queries to resolve.

[in] pDestinationBuffer

Type: [ID3D12Resource*](#)

Specifies an [ID3D12Resource](#) destination buffer, which must be in the state [D3D12_RESOURCE_STATE_COPY_DEST](#).

[in] AlignedDestinationBufferOffset

Type: [UINT64](#)

Specifies an alignment offset into the destination buffer. Must be a multiple of 8 bytes.

Return value

None

Remarks

ResolveQueryData performs a batched operation that writes query data into a destination buffer. Query data is written contiguously to the destination buffer, and the parameter.

ResolveQueryData turns application-opaque query data in an application-opaque query heap into adapter-agnostic values usable by your application. Resolving queries within a heap that have not been completed (so have had

[ID3D12GraphicsCommandList::BeginQuery](#) called for them, but not [ID3D12GraphicsCommandList::EndQuery](#)), or that have been uninitialized, results in undefined behavior and might cause device hangs or removal. The debug layer will emit an error if it detects an application has resolved incomplete or uninitialized queries.

ⓘ Note

Resolving incomplete or uninitialized queries is undefined behavior because the driver might internally store GPUVAs or other data within unresolved queries. And so attempting to resolve these queries on uninitialized data could cause a page fault or device hang. Older versions of the debug layer didn't validate this behavior.

Binary occlusion queries write 64-bits per query. The least significant bit is either 0 (the object was entirely occluded) or 1 (at least 1 sample of the object would have been

drawn). The rest of the bits are 0. Occlusion queries write 64-bits per query. The value is the number of samples that passed testing. Timestamp queries write 64-bits per query, which is a tick value that must be compared to the respective command queue frequency (see [Timing](#)).

Pipeline statistics queries write a [D3D12_QUERY_DATA_PIPELINE_STATISTICS](#) structure per query. All stream-out statistics queries write a [D3D12_QUERY_DATA_SO_STATISTICS](#) structure per query.

The core runtime will validate the following.

- *StartIndex* and *NumQueries* are within range.
- *AlignedDestinationBufferOffset* is a multiple of 8 bytes.
- *DestinationBuffer* is a buffer.
- The written data will not overflow the output buffer.
- The query type must be supported by the command list type.
- The query type must be supported by the query heap.

The debug layer will issue a warning if the destination buffer is not in the [D3D12_RESOURCE_STATE_COPY_DEST](#) state, or if any queries being resolved have not had [ID3D12GraphicsCommandList::EndQuery](#) called on them.

Examples

The [D3D12PredicationQueries](#) sample uses [ID3D12GraphicsCommandList::ResolveQueryData](#) as follows:

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12PredicationQueries::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
        // re-recording.
    ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
```

```

ID3D12DescriptorHeap* ppHeaps[] = { m_cbvHeap.Get() };
m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

m_commandList->RSSetViewports(1, &m_viewport);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList->ClearDepthStencilView(dsvHandle, D3D12_CLEAR_FLAG_DEPTH,
1.0f, 0, 0, nullptr);

// Draw the quads and perform the occlusion query.
{
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvFarQuad(m_cbvHeap-
>GetGPUDescriptorHandleForHeapStart(), m_frameIndex * CbvCountPerFrame,
m_cbvSrvDescriptorSize);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvNearQuad(cbvFarQuad,
m_cbvSrvDescriptorSize);

    m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);

        // Draw the far quad conditionally based on the result of the
        // occlusion query
        // from the previous frame.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPredication(m_queryResult.Get(), 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->DrawInstanced(4, 1, 0, 0);

        // Disable predication and always draw the near quad.
        m_commandList->SetPredication(nullptr, 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvNearQuad);
        m_commandList->DrawInstanced(4, 1, 4, 0);

        // Run the occlusion query with the bounding box quad.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPipelineState(m_queryState.Get());
        m_commandList->BeginQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);
        m_commandList->DrawInstanced(4, 1, 8, 0);
}

```

```

        m_commandList->EndQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);

        // Resolve the occlusion query and store the results in the query
result buffer
        // to be used on the subsequent frame.
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_PREDICATION, D3D12_RESOURCE_STATE_COPY_DEST));
        m_commandList->ResolveQueryData(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0, 1, m_queryResult.Get(), 0);
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_PREDICATION));
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::ResolveSubresource method (d3d12.h)

Article 08/03/2021

Copy a multi-sampled resource into a non-multi-sampled resource.

Syntax

C++

```
void ResolveSubresource(
    ID3D12Resource *pDstResource,
    UINT             DstSubresource,
    ID3D12Resource *pSrcResource,
    UINT             SrcSubresource,
    DXGI_FORMAT     Format
);
```

Parameters

pDstResource

Type: [in] [ID3D12Resource*](#)

Destination resource. Must be created on a [D3D12_HEAP_TYPE_DEFAULT](#) heap and be single-sampled. See [ID3D12Resource](#).

DstSubresource

Type: [in] [UINT](#)

A zero-based index, that identifies the destination subresource. Use [D3D12CalcSubresource](#) to calculate the subresource index if the parent resource is complex.

pSrcResource

Type: [in] [ID3D12Resource*](#)

Source resource. Must be multisampled.

SrcSubresource

Type: [in] **UINT**

The source subresource of the source resource.

Format

Type: [in] **DXGI_FORMAT**

A [DXGI_FORMAT](#) that indicates how the multisampled resource will be resolved to a single-sampled resource. See remarks.

Return value

None

Remarks

Debug layer

The debug layer will issue an error if the subresources referenced by the source view is not in the [D3D12_RESOURCE_STATE_RESOLVE_SOURCE](#) state.

The debug layer will issue an error if the destination buffer is not in the [D3D12_RESOURCE_STATE_RESOLVE_DEST](#) state.

The source and destination resources must be the same resource type and have the same dimensions. In addition, they must have compatible formats. There are three scenarios for this:

[+] Expand table

Scenario	Requirements
Source and destination are prestructured and typed	Both the source and destination must have identical formats and that format must be specified in the Format parameter.
One resource is prestructured and typed and the other is prestructured and typeless	The typed resource must have a format that is compatible with the typeless resource (i.e. the typed resource is DXGI_FORMAT_R32_FLOAT and the typeless resource is DXGI_FORMAT_R32_TYPELESS). The format of the typed resource must be specified in the Format parameter.

Source and destination are prestructured and typeless	Both the source and destination must have the same typeless format (i.e. both must have DXGI_FORMAT_R32_TYPELESS), and the Format parameter must specify a format that is compatible with the source and destination (i.e. if both are DXGI_FORMAT_R32_TYPELESS then DXGI_FORMAT_R32_FLOAT could be specified in the Format parameter). For example, given the DXGI_FORMAT_R16G16B16A16_TYPELESS format: <ul style="list-style-type: none">• The source (or dest) format could be DXGI_FORMAT_R16G16B16A16_UNORM• The dest (or source) format could be DXGI_FORMAT_R16G16B16A16_FLOAT
---	--

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Subresources](#)

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::Resource Barrier method (d3d12.h)

Article 07/27/2022

Notifies the driver that it needs to synchronize multiple accesses to resources.

Syntax

C++

```
void ResourceBarrier(
    [in] UINT NumBarriers,
    [in] const D3D12_RESOURCE_BARRIER *pBarriers
);
```

Parameters

[in] NumBarriers

Type: **UINT**

The number of submitted barrier descriptions.

[in] pBarriers

Type: **const D3D12_RESOURCE_BARRIER***

Pointer to an array of barrier descriptions.

Return value

None

Remarks

ⓘ Note

A resource to be used for the

[D3D12 RESOURCE STATE RAYTRACING ACCELERATION STRUCTURE](#) state must

be created in that state, and then never transitioned out of it. Nor may a resource that was created not in that state be transitioned into it. For more info, see [Acceleration structure memory restrictions](#) in the DirectX raytracing (DXR) functional specification on GitHub.

There are three types of barrier descriptions:

- [D3D12_RESOURCE_TRANSITION_BARRIER](#) - Transition barriers indicate that a set of subresources transition between different usages. The caller must specify the *before* and *after* usages of the subresources. The D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES flag is used to transition all subresources in a resource at the same time.
- [D3D12_RESOURCE_ALIASING_BARRIER](#) - Aliasing barriers indicate a transition between usages of two different resources which have mappings into the same heap. The application can specify both the before and the after resource. Note that one or both resources can be NULL (indicating that any tiled resource could cause aliasing).
- [D3D12_RESOURCE_UAV_BARRIER](#) - Unordered access view barriers indicate all UAV accesses (read or writes) to a particular resource must complete before any future UAV accesses (read or write) can begin. The specified resource may be NULL. It is not necessary to insert a UAV barrier between two draw or dispatch calls which only read a UAV. Additionally, it is not necessary to insert a UAV barrier between two draw or dispatch calls which write to the same UAV if the application knows that it is safe to execute the UAV accesses in any order. The resource can be NULL (indicating that any UAV access could require the barrier).

When `ID3D12GraphicsCommandList::ResourceBarrier` is passed an array of resource barrier descriptions, the API behaves as if it was called N times (1 for each array element), in the specified order. Transitions should be batched together into a single API call when possible, as a performance optimization.

For descriptions of the usage states a subresource can be in, see the [D3D12_RESOURCE_STATES](#) enumeration and the [Using Resource Barriers to Synchronize Resource States in Direct3D 12](#) section.

All subresources in a resource must be in the RENDER_TARGET state, or DEPTH_WRITE state, for render targets/depth-stencil resources respectively, when `ID3D12GraphicsCommandList::DiscardResource` is called.

When a back buffer is presented, it must be in the D3D12_RESOURCE_STATE_PRESENT state. If `IDXGISwapChain1::Present1` is called on a resource which is not in the PRESENT state, a debug layer warning will be emitted.

The resource usage bits are group into two categories, read-only and read/write.

The following usage bits are read-only:

- D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER
- D3D12_RESOURCE_STATE_INDEX_BUFFER
- D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE
- D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE
- D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT
- D3D12_RESOURCE_STATE_COPY_SOURCE
- D3D12_RESOURCE_STATE_DEPTH_READ

The following usage bits are read/write:

- D3D12_RESOURCE_STATE_UNORDERED_ACCESS
- D3D12_RESOURCE_STATE_DEPTH_WRITE

The following usage bits are write-only:

- D3D12_RESOURCE_STATE_COPY_DEST
- D3D12_RESOURCE_STATE_RENDER_TARGET
- D3D12_RESOURCE_STATE_STREAM_OUT

At most one write bit can be set. If any write bit is set, then no read bit may be set. If no write bit is set, then any number of read bits may be set.

At any given time, a subresource is in exactly one state (determined by a set of flags). The application must ensure that the states are matched when making a sequence of **ResourceBarrier** calls. In other words, the before and after states in consecutive calls to **ResourceBarrier** must agree.

To transition all subresources within a resource, the application can set the subresource index to D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES, which implies that all subresources are changed.

For improved performance, applications should use split barriers (refer to [Multi-engine synchronization](#)). Your application should also batch multiple transitions into a single call whenever possible.

Runtime validation

The runtime will validate that the barrier type values are valid members of the [D3D12_RESOURCE_BARRIER_TYPE](#) enumeration.

In addition, the runtime checks the following:

- The resource pointer is non-NULL.
- The subresource index is valid
- The before and after states are supported by the [D3D12_RESOURCE_BINDING_TIER](#) and [D3D12_RESOURCE_FLAGS](#) flags of the resource.
- Reserved bits in the state masks are not set.
- The before and after states are different.
- The set of bits in the before and after states are valid.
- If the D3D12_RESOURCE_STATE_RESOLVE_SOURCE bit is set, then the resource sample count must be greater than 1.
- If the D3D12_RESOURCE_STATE_RESOLVE_DEST bit is set, then the resource sample count must be equal to 1.

For aliasing barriers the runtime will validate that, if either resource pointer is non-NULL, it refers to a tiled resource.

For UAV barriers the runtime will validate that, if the resource is non-NULL, the resource has the D3D12_RESOURCE_STATE_UNORDERED_ACCESS bind flag set.

Validation failure causes [ID3D12GraphicsCommandList::Close](#) to return E_INVALIDARG.

Debug layer

The debug layer normally issues errors where runtime validation fails:

- If a subresource transition in a command list is inconsistent with previous transitions in the same command list.
- If a resource is used without first calling **ResourceBarrier** to put the resource into the correct state.
- If a resource is illegally bound for read and write at the same time.
- If the *before* states passed to the **ResourceBarrier** do not match the *after* states of previous calls to **ResourceBarrier**, including the aliasing case.

Whereas the debug layer attempts to validate the runtime rules, it operates conservatively so that debug layer errors are real errors, and in some cases real errors may not produce debug layer errors.

The debug layer will issue warnings in the following cases:

- All of the cases where the D3D12 debug layer would issue warnings for [ID3D12GraphicsCommandList::ResourceBarrier](#).
- If a depth buffer is used in a non-read-only mode while the resource has the D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE usage bit set.

Examples

The [D3D12HelloTriangle](#) sample uses `ID3D12GraphicsCommandList::ResourceBarrier` as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
    // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);
```

```

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

ThrowIfFailed(m_commandList->Close());
}

```

See Example Code in the D3D12 Reference.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::RSSetScissorRects method (d3d12.h)

Article 02/22/2024

Binds an array of scissor rectangles to the rasterizer stage.

Syntax

C++

```
void RSSetScissorRects(  
    [in] UINT             NumRects,  
    [in] const D3D12_RECT *pRects  
>;
```

Parameters

[in] NumRects

Type: **UINT**

The number of scissor rectangles to bind.

[in] pRects

Type: **const D3D12_RECT***

An array of scissor rectangles.

Return value

None

Remarks

All scissor rectangles must be set atomically as one operation. Any scissor rectangles not defined by the call are disabled.

Which scissor rectangle to use is determined by the **SV_ViewportArrayIndex** semantic output by a geometry shader (see shader semantic syntax). If a geometry shader does

not make use of the `SV_ViewportArrayIndex` semantic then Direct3D will use the first scissor rectangle in the array.

Each scissor rectangle in the array corresponds to a viewport in an array of viewports (see [RSSetViewports](#)).

Examples

The [D3D12HelloTriangle](#) sample uses `ID3D12GraphicsCommandList::RSSetScissorRects` as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
// Command list allocators can only be reset when the associated
// command lists have finished execution on the GPU; apps should use
// fences to determine GPU execution progress.
ThrowIfFailed(m_commandAllocator->Reset());

// However, when ExecuteCommandList() is called on a particular command
// list, that command list can then be reset at any time and must be before
// re-recording.
ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
m_pipelineState.Get()));

// Set necessary state.
m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
m_commandList->RSSetViewports(1, &m_viewport);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));
```

```

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList->IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

ThrowIfFailed(m_commandList->Close());

```

See Example Code in the [D3D12 Reference](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::RSSetViewports method (d3d12.h)

Article 02/22/2024

Bind an array of viewports to the rasterizer stage of the pipeline.

Syntax

C++

```
void RSSetViewports(  
    [in] UINT             NumViewports,  
    [in] const D3D12_VIEWPORT *pViewports  
)
```

Parameters

[in] NumViewports

Type: **UINT**

Number of viewports to bind. The range of valid values is (0, D3D12_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE).

[in] pViewports

Type: **const D3D12_VIEWPORT***

An array of **D3D12_VIEWPORT** structures to bind to the device.

Return value

None

Remarks

All viewports must be set atomically as one operation. Any viewports not defined by the call are disabled.

Which viewport to use is determined by the [SV_ViewportArrayIndex](#) semantic output by a geometry shader; if a geometry shader does not specify the semantic, Direct3D will use the first viewport in the array.

Examples

The [D3D12HelloTriangle](#) sample uses [ID3D12GraphicsCommandList::RSSetViewports](#) as follows:

C++

```
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
UINT m_rtvDescriptorSize;
```

C++

```
void D3D12HelloTriangle::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
        // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_commandAllocator.Get(),
m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());
    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));
```

```

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, nullptr);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);
m_commandList->DrawInstanced(3, 1, 0, 0);

// Indicate that the back buffer will now be used to present.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

ThrowIfFailed(m_commandList->Close());
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::SetComputeRoot32BitConstant method (d3d12.h)

Article 02/22/2024

Sets a constant in the compute root signature.

Syntax

C++

```
void SetComputeRoot32BitConstant(
    [in] UINT RootParameterIndex,
    [in] UINT SrcData,
    [in] UINT DestOffsetIn32BitValues
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] SrcData

Type: **UINT**

The source data for the constant to set.

[in] DestOffsetIn32BitValues

Type: **UINT**

The offset, in 32-bit values, to set the constant in the root signature.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::SetComputeRoot32BitConstants method (d3d12.h)

Article 02/22/2024

Sets a group of constants in the compute root signature.

Syntax

C++

```
void SetComputeRoot32BitConstants(
    [in] UINT          RootParameterIndex,
    [in] UINT          Num32BitValuesToSet,
    [in] const void*   pSrcData,
    [in] UINT          DestOffsetIn32BitValues
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] Num32BitValuesToSet

Type: **UINT**

The number of constants to set in the root signature.

[in] pSrcData

Type: **const void***

The source data for the group of constants to set.

[in] DestOffsetIn32BitValues

Type: **UINT**

The offset, in 32-bit values, to set the first constant of the group in the root signature.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetComputeRootConstantBufferView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the constant buffer in the compute root signature.

Syntax

C++

```
void SetComputeRootConstantBufferView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

Specifies the D3D12_GPU_VIRTUAL_ADDRESS of the constant buffer.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetComputeRootDescriptorTable method (d3d12.h)

Article 02/22/2024

Sets a descriptor table into the compute root signature.

Syntax

C++

```
void SetComputeRootDescriptorTable(  
    [in] UINT RootParameterIndex,  
    [in] D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor  
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BaseDescriptor

Type: **D3D12_GPU_DESCRIPTOR_HANDLE**

A GPU_descriptor_handle object for the base descriptor to set.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetComputeRootShaderResourceView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the shader resource in the compute root signature.

Syntax

C++

```
void SetComputeRootShaderResourceView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

The GPU virtual address of the buffer. D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd alias of **UINT64**.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetComputeRootSignature method (d3d12.h)

Article 02/22/2024

Sets the layout of the compute root signature.

Syntax

C++

```
void SetComputeRootSignature(  
    [in, optional] ID3D12RootSignature *pRootSignature  
);
```

Parameters

[in, optional] pRootSignature

Type: [ID3D12RootSignature*](#)

A pointer to the [ID3D12RootSignature](#) object.

Return value

None

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetComputeRootUnorderedAccessView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the unordered-access-view resource in the compute root signature.

Syntax

C++

```
void SetComputeRootUnorderedAccessView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: [UINT](#)

The slot number for binding.

[in] BufferLocation

Type: [D3D12_GPU_VIRTUAL_ADDRESS](#)

The GPU virtual address of the buffer. D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd alias of [UINT64](#).

Return value

None

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::SetDescriptorHeaps method (d3d12.h)

Article04/02/2021

Changes the currently bound descriptor heaps that are associated with a command list.

Syntax

C++

```
void SetDescriptorHeaps(
    UINT             NumDescriptorHeaps,
    ID3D12DescriptorHeap * const *ppDescriptorHeaps
);
```

Parameters

NumDescriptorHeaps

Type: [in] **UINT**

Number of descriptor heaps to bind.

ppDescriptorHeaps

Type: [in] **ID3D12DescriptorHeap***

A pointer to an array of **ID3D12DescriptorHeap** objects for the heaps to set on the command list.

You can only bind descriptor heaps of type
D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV and
D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER.

Only one descriptor heap of each type can be set at one time, which means a maximum of 2 heaps (one sampler, one CBV/SRV/UAV) can be set at one time.

Return value

None

Remarks

`SetDescriptorHeaps` can be called on a bundle, but the bundle descriptor heaps must match the calling command list descriptor heap. For more information on bundle restrictions, refer to [Creating and Recording Command Lists and Bundles](#).

All previously set heaps are unset by the call. At most one heap of each shader-visible type can be set in the call.

Changing descriptor heaps can incur a pipeline flush on some hardware. Because of this, it is recommended to use a single shader-visible heap of each type, and set it once per frame, rather than regularly changing the bound descriptor heaps. Instead, use [`ID3D12Device::CopyDescriptors`](#) and [`ID3D12Device::CopyDescriptorsSimple`](#) to copy the required descriptors from shader-opaque heaps to the single shader-visible heap as required during rendering.

Examples

The [D3D12Bundles](#) sample uses [`ID3D12GraphicsCommandList::SetDescriptorHeaps`](#) as follows:

C++

```
void D3D12Bundles::PopulateCommandList(FrameResource* pFrameResource)
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_pCurrentFrameResource->m_commandAllocator->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    before
    // re-recording.
    ThrowIfFailed(m_commandList->Reset(m_pCurrentFrameResource-
        >m_commandAllocator.Get(), m_pipelineState1.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvSrvHeap.Get(),
        m_samplerHeap.Get() };
    m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

    m_commandList->RSSetViewports(1, &m_viewport);
    m_commandList->RSSetScissorRects(1, &m_scissorRect);

    // Indicate that the back buffer will be used as a render target.
```

```

    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

    CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
    CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
    m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

    // Record commands.
    const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
    m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
    m_commandList->ClearDepthStencilView(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart(), D3D12_CLEAR_FLAG_DEPTH, 1.0f, 0, 0,
nullptr);

    if (UseBundles)
    {
        // Execute the prebuilt bundle.
        m_commandList->ExecuteBundle(pFrameResource->m_bundle.Get());
    }
    else
    {
        // Populate a new command list.
        pFrameResource->PopulateCommandList(m_commandList.Get(),
m_pipelineState1.Get(), m_pipelineState2.Get(), m_currentFrameResourceIndex,
m_numIndices, &m_indexBufferView,
&m_vertexBufferView, m_cbvSrvHeap.Get(), m_cbvSrvDescriptorSize,
m_samplerHeap.Get(), m_rootSignature.Get());
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

[See Example Code in the D3D12 Reference.](#)

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3d12.lib
DLL	D3d12.dll

See also

[Descriptor Heaps](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant method (d3d12.h)

Article 02/22/2024

Sets a constant in the graphics root signature.

Syntax

C++

```
void SetGraphicsRoot32BitConstant(  
    [in] UINT RootParameterIndex,  
    [in] UINT SrcData,  
    [in] UINT DestOffsetIn32BitValues  
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] SrcData

Type: **UINT**

The source data for the constant to set.

[in] DestOffsetIn32BitValues

Type: **UINT**

The offset, in 32-bit values, to set the constant in the root signature.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants method (d3d12.h)

Article 02/22/2024

Sets a group of constants in the graphics root signature.

Syntax

C++

```
void SetGraphicsRoot32BitConstants(
    [in] UINT          RootParameterIndex,
    [in] UINT          Num32BitValuesToSet,
    [in] const void*  pSrcData,
    [in] UINT          DestOffsetIn32BitValues
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] Num32BitValuesToSet

Type: **UINT**

The number of constants to set in the root signature.

[in] pSrcData

Type: **const void***

The source data for the group of constants to set.

[in] DestOffsetIn32BitValues

Type: **UINT**

The offset, in 32-bit values, to set the first constant of the group in the root signature.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the constant buffer in the graphics root signature.

Syntax

C++

```
void SetGraphicsRootConstantBufferView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

The GPU virtual address of the constant buffer. D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd alias of UINT64.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable method (d3d12.h)

Article 02/22/2024

Sets a descriptor table into the graphics root signature.

Syntax

C++

```
void SetGraphicsRootDescriptorTable(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor
);
```

Parameters

[in] RootParameterIndex

Type: [UINT](#)

The slot number for binding.

[in] BaseDescriptor

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A GPU_descriptor_handle object for the base descriptor to set.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the shader resource in the graphics root signature.

Syntax

C++

```
void SetGraphicsRootShaderResourceView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

The GPU virtual address of the Buffer. Textures are not supported.
D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd alias of **UINT64**.

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRootSignature method (d3d12.h)

Article 02/22/2024

Sets the layout of the graphics root signature.

Syntax

C++

```
void SetGraphicsRootSignature(  
    [in, optional] ID3D12RootSignature *pRootSignature  
);
```

Parameters

[in, optional] pRootSignature

Type: [ID3D12RootSignature*](#)

A pointer to the [ID3D12RootSignature](#) object.

Return value

None

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetGraphicsRootUnorderedAccessView method (d3d12.h)

Article 02/22/2024

Sets a CPU descriptor handle for the unordered-access-view resource in the graphics root signature.

Syntax

C++

```
void SetGraphicsRootUnorderedAccessView(
    [in] UINT RootParameterIndex,
    [in] D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

[in] RootParameterIndex

Type: **UINT**

The slot number for binding.

[in] BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

The GPU virtual address of the buffer. D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd alias of UINT64.

Return value

None

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList::SetMarker method (d3d12.h)

Article 02/22/2024

Not intended to be called directly. Use the [PIX event runtime](#) to insert events into a command list.

Syntax

C++

```
void SetMarker(
    UINT      Metadata,
    [in, optional] const void *pData,
    UINT      Size
);
```

Parameters

Metadata

Type: **UINT**

Internal.

[in, optional] pData

Type: **const void***

Internal.

Size

Type: **UINT**

Internal.

Return value

None

Remarks

This is a support method used internally by the PIX event runtime. It is not intended to be called directly.

To insert instrumentation markers at the current location within a D3D12 command list, use the **PIXSetMarker** function. This is provided by the [WinPixEventRuntime](#) NuGet package.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?



[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetPipelineState method (d3d12.h)

Article 02/22/2024

Sets all shaders and programs most of the fixed-function state of the graphics processing unit (GPU) pipeline.

Syntax

C++

```
void SetPipelineState(  
    [in] ID3D12PipelineState *pPipelineState  
);
```

Parameters

[in] pPipelineState

Type: [ID3D12PipelineState*](#)

Pointer to the [ID3D12PipelineState](#) containing the pipeline state data.

Return value

None

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SetPrediction method (d3d12.h)

Article 02/22/2024

Sets a rendering predicate.

Syntax

C++

```
void SetPrediction(
    [in, optional] ID3D12Resource      *pBuffer,
    [in]          UINT64                AlignedBufferOffset,
    [in]          D3D12_PREDICATION_OP Operation
);
```

Parameters

[in, optional] pBuffer

Type: [ID3D12Resource*](#)

The buffer, as an [ID3D12Resource](#), which must be in the [D3D12_RESOURCE_STATE_PREDICATION](#) or [D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT](#) state (both values are identical, and provided as aliases for clarity), or **NULL** to disable predication.

[in] AlignedBufferOffset

Type: [UINT64](#)

The aligned buffer offset, as a [UINT64](#).

[in] Operation

Type: [D3D12_PREDICATION_OP](#)

Specifies a [D3D12_PREDICATION_OP](#), such as [D3D12_PREDICATION_OP_EQUAL_ZERO](#) or [D3D12_PREDICATION_OP_NOT_EQUAL_ZERO](#).

Return value

None

Remarks

Use this method to denote that subsequent rendering and resource manipulation commands are not actually performed if the resulting predicate data of the predicate is equal to the operation specified.

Unlike Direct3D 11, in Direct3D 12 predication state is not inherited by direct command lists, and predication is always respected (there are no predication hints). All direct command lists begin with predication disabled. Bundles do inherit predication state. It is legal for the same predicate to be bound multiple times.

Illegal API calls will result in [Close](#) returning an error, or [ID3D12CommandQueue::ExecuteCommandLists](#) dropping the command list and removing the device.

The debug layer will issue errors whenever the runtime validation fails.

Refer to [Predication](#) for more information.

Examples

The [D3D12PredicationQueries](#) sample uses [ID3D12GraphicsCommandList::SetPredication](#) as follows:

C++

```
// Fill the command list with all the render commands and dependent state.
void D3D12PredicationQueries::PopulateCommandList()
{
    // Command list allocators can only be reset when the associated
    // command lists have finished execution on the GPU; apps should use
    // fences to determine GPU execution progress.
    ThrowIfFailed(m_commandAllocators[m_frameIndex]->Reset());

    // However, when ExecuteCommandList() is called on a particular command
    // list, that command list can then be reset at any time and must be
    // before
    // re-recording.
    ThrowIfFailed(m_commandList-
>Reset(m_commandAllocators[m_frameIndex].Get(), m_pipelineState.Get()));

    // Set necessary state.
    m_commandList->SetGraphicsRootSignature(m_rootSignature.Get());

    ID3D12DescriptorHeap* ppHeaps[] = { m_cbvHeap.Get() };
}
```

```

m_commandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);

m_commandList->RSSetViewports(1, &m_viewport);
m_commandList->RSSetScissorRects(1, &m_scissorRect);

// Indicate that the back buffer will be used as a render target.
m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_PRESENT, D3D12_RESOURCE_STATE_RENDER_TARGET));

CD3DX12_CPU_DESCRIPTOR_HANDLE rtvHandle(m_rtvHeap-
>GetCPUDescriptorHandleForHeapStart(), m_frameIndex, m_rtvDescriptorSize);
CD3DX12_CPU_DESCRIPTOR_HANDLE dsvHandle(m_dsvHeap-
>GetCPUDescriptorHandleForHeapStart());
m_commandList->OMSetRenderTargets(1, &rtvHandle, FALSE, &dsvHandle);

// Record commands.
const float clearColor[] = { 0.0f, 0.2f, 0.4f, 1.0f };
m_commandList->ClearRenderTargetView(rtvHandle, clearColor, 0, nullptr);
m_commandList->ClearDepthStencilView(dsvHandle, D3D12_CLEAR_FLAG_DEPTH,
1.0f, 0, 0, nullptr);

// Draw the quads and perform the occlusion query.
{
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvFarQuad(m_cbvHeap-
>GetGPUDescriptorHandleForHeapStart(), m_frameIndex * CbvCountPerFrame,
m_cbvSrvDescriptorSize);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvNearQuad(cbvFarQuad,
m_cbvSrvDescriptorSize);

    m_commandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP);
    m_commandList->IASetVertexBuffers(0, 1, &m_vertexBufferView);

        // Draw the far quad conditionally based on the result of the
        // occlusion query
        // from the previous frame.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPredication(m_queryResult.Get(), 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->DrawInstanced(4, 1, 0, 0);

        // Disable predication and always draw the near quad.
        m_commandList->SetPredication(nullptr, 0,
D3D12_PREDICATION_OP_EQUAL_ZERO);
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvNearQuad);
        m_commandList->DrawInstanced(4, 1, 4, 0);

        // Run the occlusion query with the bounding box quad.
        m_commandList->SetGraphicsRootDescriptorTable(0, cbvFarQuad);
        m_commandList->SetPipelineState(m_queryState.Get());
        m_commandList->BeginQuery(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);
        m_commandList->DrawInstanced(4, 1, 8, 0);
        m_commandList->EndQuery(m_queryHeap.Get()),

```

```

D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0);

        // Resolve the occlusion query and store the results in the query
        result buffer
        // to be used on the subsequent frame.
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_PREDICATION, D3D12_RESOURCE_STATE_COPY_DEST));
        m_commandList->ResolveQueryData(m_queryHeap.Get(),
D3D12_QUERY_TYPE_BINARY_OCCLUSION, 0, 1, m_queryResult.Get(), 0);
        m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_queryResult.Get(),
D3D12_RESOURCE_STATE_COPY_DEST, D3D12_RESOURCE_STATE_PREDICATION));
    }

    // Indicate that the back buffer will now be used to present.
    m_commandList->ResourceBarrier(1,
&CD3DX12_RESOURCE_BARRIER::Transition(m_renderTargets[m_frameIndex].Get(),
D3D12_RESOURCE_STATE_RENDER_TARGET, D3D12_RESOURCE_STATE_PRESENT));

    ThrowIfFailed(m_commandList->Close());
}

```

See Example Code in the D3D12 Reference.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Predication queries walk-through](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList::SOSetTargets method (d3d12.h)

Article 02/22/2024

Sets the stream output buffer views.

Syntax

C++

```
void SOSetTargets(
    [in]          UINT           StartSlot,
    [in]          UINT           NumViews,
    [in, optional] const D3D12_STREAM_OUTPUT_BUFFER_VIEW *pViews
);
```

Parameters

[in] StartSlot

Type: **UINT**

Index into the device's zero-based array to begin setting stream output buffers.

[in] NumViews

Type: **UINT**

The number of entries in the *pViews* array.

[in, optional] pViews

Type: **const D3D12_STREAM_OUTPUT_BUFFER_VIEW***

Specifies an array of **D3D12_STREAM_OUTPUT_BUFFER_VIEW** structures.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList1 interface (d3d12.h)

Article 07/22/2021

Encapsulates a list of graphics commands for rendering, extending the interface to support programmable sample positions, atomic copies for implementing late-latch techniques, and optional depth-bounds testing.

Note This interface, introduced in the Windows 10 Creators Update, is the latest version of the [ID3D12GraphicsCommandList](#) interface. Applications targetting Windows 10 Creators Update should use this interface instead of [ID3D12GraphicsCommandList](#).

Inheritance

The [ID3D12GraphicsCommandList1](#) interface inherits from [ID3D12GraphicsCommandList](#). [ID3D12GraphicsCommandList1](#) also has these types of members:

Methods

The [ID3D12GraphicsCommandList1](#) interface has these methods.

[+] Expand table

[ID3D12GraphicsCommandList1::AtomicCopyBufferUINT](#)

Atomically copies a primary data element of type `UINT` from one resource to another, along with optional dependent resources.

[ID3D12GraphicsCommandList1::AtomicCopyBufferUINT64](#)

Atomically copies a primary data element of type `UINT64` from one resource to another, along with optional dependent resources.

[ID3D12GraphicsCommandList1::OMSetDepthBounds](#)

This method enables you to change the depth bounds dynamically.

[ID3D12GraphicsCommandList1::ResolveSubresourceRegion](#)

Copy a region of a multisampled or compressed resource into a non-multisampled or non-compressed resource.

[ID3D12GraphicsCommandList1::SetSamplePositions](#)

This method configures the sample positions used by subsequent draw, copy, resolve, and similar operations.

[ID3D12GraphicsCommandList1::SetViewInstanceMask](#)

Set a mask that controls which view instances are enabled for subsequent draws.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12GraphicsCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::AtomicCopyBufferUINT method (d3d12.h)

Article 08/19/2022

Atomically copies a primary data element of type UINT from one resource to another, along with optional dependent resources.

These 'dependent resources' are so-named because they depend upon the primary data element to locate them, typically the key element is an address, index, or other handle that refers to one or more the dependent resources indirectly.

This function supports a primary data element of type UINT (32bit). A different version of this function, [AtomicCopyBufferUINT64](#), supports a primary data element of type UINT64 (64bit).

Syntax

C++

```
void AtomicCopyBufferUINT(
    [in] ID3D12Resource                      *pDstBuffer,
    UINT64                                     DstOffset,
    [in] ID3D12Resource                      *pSrcBuffer,
    UINT64                                     SrcOffset,
    UINT                                       Dependencies,
    [in] ID3D12Resource                      * const *ppDependentResources,
    [in] const D3D12_SUBRESOURCE_RANGE_UINT64 *pDependentSubresourceRanges
);
```

Parameters

[in] pDstBuffer

Type: **ID3D12Resource***

SAL: *In*

The resource that the UINT primary data element is copied into.

DstOffset

Type: **UINT64**

An offset into the destination resource buffer that specifies where the primary data element is copied into, in bytes. This offset combined with the base address of the resource buffer must result in a memory address that's naturally aligned for UINT values.

[in] pSrcBuffer

Type: ID3D12Resource*

SAL: *In*

The resource that the UINT primary data element is copied from. This data is typically an address, index, or other handle that shader code can use to locate the most-recent version of latency-sensitive information.

SrcOffset

Type: **UINT64**

An offset into the source resource buffer that specifies where the primary data element is copied from, in bytes. This offset combined with the base address of the resource buffer must result in a memory address that's naturally aligned for UINT values.

Dependencies

Type: **UINT**

The number of dependent resources.

[in] ppDependentResources

Type: ID3D12Resource*

SAL: *In_reads(Dependencies)*

An array of resources that contain the dependent elements of the data payload.

[in] pDependentSubresourceRanges

Type: const **D3D12_SUBRESOURCE_RANGE_UINT64***

SAL: *In_reads(Dependencies)*

An array of subresource ranges that specify the dependent elements of the data payload. These elements are completely updated before the primary data element is itself atomically copied. This ensures that the entire operation is logically atomic; that is, the primary data element never refers to an incomplete data payload.

Return value

None

Remarks

This method is typically used to update resources for which normal rendering pipeline latency can be detrimental to user experience. For example, an application can compute a view matrix from the latest user input (such as from the sensors of a head-mounted display), and use this function to update and activate this matrix in command lists already dispatched to the GPU to reduce perceived latency between input and rendering.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::AtomicCopyBufferUINT64 method (d3d12.h)

Article 02/22/2024

Atomically copies a primary data element of type UINT64 from one resource to another, along with optional dependent resources.

These 'dependent resources' are so-named because they depend upon the primary data element to locate them, typically the key element is an address, index, or other handle that refers to one or more the dependent resources indirectly.

This function supports a primary data element of type UINT64 (64bit). A different version of this function, [AtomicCopyBufferUINT](#), supports a primary data element of type UINT (32bit).

Syntax

C++

```
void AtomicCopyBufferUINT64(
    [in] ID3D12Resource                  *pDstBuffer,
    UINT64                               DstOffset,
    [in] ID3D12Resource                  *pSrcBuffer,
    UINT64                               SrcOffset,
    UINT                                 Dependencies,
    [in] ID3D12Resource                  * const *ppDependentResources,
    [in] const D3D12_SUBRESOURCE_RANGE_UINT64 *pDependentSubresourceRanges
);
```

Parameters

[in] pDstBuffer

Type: **ID3D12Resource***

SAL: *In*

The resource that the UINT64 primary data element is copied into.

DstOffset

Type: **UINT64**

An offset into the destination resource buffer that specifies where the primary data element is copied into, in bytes. This offset combined with the base address of the resource buffer must result in a memory address that's naturally aligned for **UINT64** values.

[in] `pSrcBuffer`

Type: **ID3D12Resource***

SAL: `In`

The resource that the **UINT64** primary data element is copied from. This data is typically an address, index, or other handle that shader code can use to locate the most-recent version of latency-sensitive information.

`SrcOffset`

Type: **UINT64**

An offset into the source resource buffer that specifies where the primary data element is copied from, in bytes. This offset combined with the base address of the resource buffer must result in a memory address that's naturally aligned for **UINT64** values.

Dependencies

Type: **UINT**

The number of dependent resources.

[in] `ppDependentResources`

Type: **ID3D12Resource***

SAL: `In_reads(Dependencies)`

An array of resources that contain the dependent elements of the data payload.

[in] `pDependentSubresourceRanges`

Type: **const D3D12_SUBRESOURCE_RANGE_UINT64***

SAL: `In_reads(Dependencies)`

An array of subresource ranges that specify the dependent elements of the data payload. These elements are completely updated before the primary data element is itself atomically copied. This ensures that the entire operation is logically atomic; that is, the primary data element never refers to an incomplete data payload.

Return value

None

Remarks

This method is typically used to update resources for which normal rendering pipeline latency can be detrimental to user experience. For example, an application can compute a view matrix from the latest user input (such as from the sensors of a head-mounted display), and use this function to update and activate this matrix in command lists already dispatched to the GPU to reduce perceived latency between input and rendering.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::OMSetDepthBounds method (d3d12.h)

Article 02/22/2024

This method enables you to change the depth bounds dynamically.

Syntax

C++

```
void OMSetDepthBounds(
    [in] FLOAT Min,
    [in] FLOAT Max
);
```

Parameters

[in] Min

Type: **FLOAT**

SAL: *In*

Specifies the minimum depth bounds. The default value is 0. NaN values silently convert to 0.

[in] Max

Type: **FLOAT**

SAL: *In*

Specifies the maximum depth bounds. The default value is 1. NaN values silently convert to 0.

Return value

None

Remarks

Depth-bounds testing allows pixels and samples to be discarded if the currently-stored depth value is outside the range specified by *Min* and *Max*, inclusive. If the currently-stored depth value of the pixel or sample is inside this range, then the depth-bounds test passes and it is rendered; otherwise, the depth-bounds test fails and the pixel or sample is discarded. Note that the depth-bounds test considers the currently-stored depth value, not the depth value generated by the executing pixel shader.

To use depth-bounds testing, the application must use the new [CreatePipelineState](#) method to enable depth-bounds testing on the PSO and then can use this command list method to change the depth-bounds dynamically.

OMSetDepthBounds is an optional feature. Use the [CheckFeatureSupport](#) method to determine whether or not this feature is supported by the user-mode driver. Support for this feature is reported through the [D3D12_FEATURE_D3D12_OPTIONS2](#) structure.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::ResolveSubresourceRegion method (d3d12.h)

Article 08/19/2022

Copy a region of a multisampled or compressed resource into a non-multisampled or non-compressed resource.

Syntax

C++

```
void ResolveSubresourceRegion(
    [in]          ID3D12Resource      *pDstResource,
    [in]          UINT                DstSubresource,
    [in]          UINT                DstX,
    [in]          UINT                DstY,
    [in]          ID3D12Resource      *pSrcResource,
    [in]          UINT                SrcSubresource,
    [in, optional] D3D12_RECT        *pSrcRect,
    [in]          DXGI_FORMAT         Format,
    [in]          D3D12_RESOLVE_MODE   ResolveMode
);
```

Parameters

[in] pDstResource

Type: ID3D12Resource*

SAL: *In*

Destination resource. Must be created with the D3D11_USAGE_DEFAULT flag and must be single-sampled unless its to be resolved from a compressed resource (D3D12_RESOLVE_MODE_DECOMPRESS); in this case it must have the same sample count as the compressed source.

[in] DstSubresource

Type: UINT

SAL: *In*

A zero-based index that identifies the destination subresource. Use [D3D12CalcSubresource](#) to calculate the subresource index if the parent resource is complex.

[in] DstX

Type: **UINT**

SAL: *In*

The X coordinate of the left-most edge of the destination region. The width of the destination region is the same as the width of the source rect.

[in] DstY

Type: **UINT**

SAL: *In*

The Y coordinate of the top-most edge of the destination region. The height of the destination region is the same as the height of the source rect.

[in] pSrcResource

Type: **ID3D12Resource***

SAL: *In*

Source resource. Must be multisampled or compressed.

[in] SrcSubresource

Type: **UINT**

SAL: *In*

A zero-based index that identifies the source subresource.

[in, optional] pSrcRect

Type: **D3D12_RECT***

SAL: *In_opt*

Specifies the rectangular region of the source resource to be resolved. Passing NULL for *pSrcRect* specifies that the entire subresource is to be resolved.

[in] Format

Type: DXGI_FORMAT

SAL: *In*

A DXGI_FORMAT that specifies how the source and destination resource formats are consolidated.

[in] ResolveMode

Type: D3D12_RESOLVE_MODE

SAL: *In*

Specifies the operation used to resolve the source samples.

When using the D3D12_RESOLVE_MODE_DECOMPRESS operation, the sample count can be larger than 1 as long as the source and destination have the same sample count, and source and destination may specify the same resource as long as the source rect aligns with the destination X and Y coordinates, in which case decompression occurs in place.

When using the D3D12_RESOLVE_MODE_MIN, D3D12_RESOLVE_MODE_MAX, or D3D12_RESOLVE_MODE_AVERAGE operation, the destination must have a sample count of 1.

Return value

None

Remarks

ResolveSubresourceRegion operates like [ResolveSubresource](#) but allows for only part of a resource to be resolved and for source samples to be resolved in several ways. Partial resolves can be useful in multi-adapter scenarios; for example, when the rendered area has been partitioned across adapters, each adapter might only need to resolve the portion of a subresource that corresponds to its assigned partition.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::SetSamplePositions method (d3d12.h)

Article 08/19/2022

This method configures the sample positions used by subsequent draw, copy, resolve, and similar operations.

Syntax

C++

```
void SetSamplePositions(
    [in] UINT             NumSamplesPerPixel,
    [in] UINT             NumPixels,
    [in] D3D12_SAMPLE_POSITION *pSamplePositions
);
```

Parameters

[in] NumSamplesPerPixel

Type: **UINT**

SAL: *In*

Specifies the number of samples to take, per pixel. This value can be 1, 2, 4, 8, or 16, otherwise the SetSamplePosition call is dropped. The number of samples must match the sample count configured in the PSO at draw time, otherwise the behavior is undefined.

[in] NumPixels

Type: **UINT**

SAL: *In*

Specifies the number of pixels that sample patterns are being specified for. This value can be either 1 or 4, otherwise the SetSamplePosition call is dropped. A value of 1 configures a single sample pattern to be used for each pixel; a value of 4 configures separate sample patterns for each pixel in a 2x2 pixel grid which is repeated over the render-target or viewport space, aligned to even coordinates.

Note that the maximum number of combined samples can't exceed 16, otherwise the call is dropped. If NumPixels is set to 4, NumSamplesPerPixel can specify no more than 4 samples.

[in] pSamplePositions

Type: [D3D12_SAMPLE_POSITION*](#)

SAL: *In_reads(NumSamplesPerPixel*NumPixels)*

Specifies an array of D3D12_SAMPLE_POSITION elements. The size of the array is NumPixels * NumSamplesPerPixel. If NumPixels is set to 4, then the first group of sample positions corresponds to the upper-left pixel in the 2x2 grid of pixels; the next group of sample positions corresponds to the upper-right pixel, the next group to the lower-left pixel, and the final group to the lower-right pixel.

If centroid interpolation is used during rendering, the order of positions for each pixel determines centroid-sampling priority. That is, the first covered sample in the order specified is chosen as the centroid sample location.

Return value

None

Remarks

The operational semantics of sample positions are determined by the various draw, copy, resolve, and other operations that can occur.

CommandList: In the absence of any prior calls to SetSamplePositions in a CommandList, samples assume the default position based on the Pipeline State Object (PSO). The default positions are determined either by the SAMPLE_DESC portion of the PSO if it is present, or by the standard sample positions if the RASTERIZER_DESC portion of the PSO has ForcedSampleCount set to a value greater than 0.

After SetSamplePosition has been called, subsequent draw calls must use a PSO that specifies a matching sample count either using the SAMPLE_DESC portion of the PSO, or ForcedSampleCount in the RASTERIZER_DESC portion of the PSO.

SetSamplePositions can only be called on a graphics CommandList. It can't be called in a bundle; bundles inherit sample position state from the calling CommandList and don't modify it.

Calling `SetSamplePositions(0, 0, NULL)` reverts the sample positions to their default values.

Clear RenderTarget: Sample positions are ignored when clearing a render target.

Clear DepthStencil: When clearing the depth portion of a depth-stencil surface or any region of it, the sample positions must be set to match those of future rendering to the cleared surface or region; the contents of any uncleared regions produced using different sample positions become undefined.

When clearing the stencil portion of a depth-stencil surface or any region of it, the sample positions are ignored.

Draw to RenderTarget: When drawing to a render target the sample positions can be changed for each draw call, even when drawing to a region that overlaps previous draw calls. The current sample positions determine the operational semantics of each draw call and samples are taken from the stored contents of the render target, even if the contents were produced using different sample positions.

Draw using DepthStencil: When drawing to a depth-stencil surface (read or write) or any region of it, the sample positions must be set to match those used to clear the affected region previously. To use a different sample position, the target region must be cleared first. The pixels outside the clear region are unaffected.

Hardware may store the depth portion or a depth-stencil surface as plane equations, and evaluate them to produce depth values when the application issues a read. Only the rasterizer and output-merger are required to support programmable sample positions of the depth portion of a depth-stencil surface. Any other read or write of the depth portion that has been rendered with sample positions set may ignore them and instead sample at the standard positions.

Resolve RenderTarget: When resolving a render target or any region of it, the sample positions are ignored; these APIs operate only on stored color values.

Resolve DepthStencil: When resolving the depth portion of a depth-stencil surface or any region of it, the sample positions must be set to match those of past rendering to the resolved surface or region. To use a different sample position, the target region must be cleared first.

When resolving the stencil portion of a depth-stencil surface or any region of it, the sample positions are ignored; stencil resolves operate only on stored stencil values.

Copy RenderTarget: When copying from a render target, the sample positions are ignored regardless of whether it is a full or partial copy.

Copy DepthStencil (Full Subresource): When copying a full subresource from a depth-stencil surface, the sample positions must be set to match the sample positions used to generate the source surface. To use a different sample position, the target region must be cleared first.

On some hardware properties of the source surface (such as stored plane equations for depth values) transfer to the destination. Therefore, if the destination surface is subsequently drawn to, the sample positions originally used to generate the source content need to be used with the destination surface. The API requires this on all hardware for consistency even if it may only apply to some.

Copy DepthStencil (Partial Subresource): When copying a partial subresource from a depth-stencil surface, the sample positions must be set to match the sample positions used to generate the source surface, similarly to copying a full subresource. However, if the content of an affected destination subresources is only partially covered by the copy, the contents of the uncovered portion within those subresources becomes undefined unless all of it was generated using the same sample positions as the copy source. To use a different sample position, the target region must be cleared first.

When copying a partial subresource from the stencil portion of a depth-stencil surface, the sample positions are ignored. It doesn't matter what sample positions were used to generate content for any other areas of the destination buffer not covered by the copy – those contents remain valid.

Shader SamplePos: The HLSL SamplePos intrinsic is not aware of programmable sample positions and results returned to shaders calling this on a surface rendered with programmable positions is undefined. Applications must pass coordinates into their shader manually if needed. Similarly evaluating attributes by sample index is undefined with programmable sample positions.

Transitioning out of DEPTH_READ or DEPTH_WRITE state: If a subresource in DEPTH_READ or DEPTH_WRITE state is transitioned to any other state, including COPY_SOURCE or RESOLVE_SOURCE, some hardware might need to decompress the surface. Therefore, the sample positions must be set on the command list to match those used to generate the content in the source surface. Furthermore, for any subsequent transitions of the surface while the same depth data remains in it, the sample positions must continue to match those set on the command list. To use a different sample position, the target region must be cleared first.

If an application wants to minimize the decompressed area when only a portion needs to be used, or just to preserve compression, ResolveSubresourceRegion() can be called in DECOMPRESS mode with a rect specified. This will decompress just the relevant area to a separate resource leaving the source intact on some hardware, though on other

hardware even the source area is decompressed. The separate explicitly decompressed resource can then be transitioned to the desired state (such as SHADER_RESOURCE).

Transitioning out of RENDER_TARGET state: If a subresource in RENDER_TARGET state is transitioned to anything other than COPY_SOURCE or RESOLVE_SOURCE, some implementations may need to decompress the surface. This decompression is agnostic to sample positions.

If an application wants to minimize the decompressed area when only a portion needs to be used, or just to preserve compression, ResolveSubresourceRegion() can be called in DECOMPRESS mode with a rect specified. This will decompress just the relevant area to a separate resource leaving the source intact on some hardware, though on other hardware even the source area is decompressed. The separate explicitly decompressed resource can then be transitioned to the desired state (such as SHADER_RESOURCE).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList1::SetViewInstanceMask method (d3d12.h)

Article02/22/2024

Set a mask that controls which view instances are enabled for subsequent draws.

Syntax

C++

```
void SetViewInstanceMask(  
    [in] UINT Mask  
);
```

Parameters

[in] Mask

Type: **UINT**

A mask that specifies which views are enabled or disabled. If bit i starting from the least-significant bit is set, view instance i is enabled.

Return value

None

Remarks

The view instance mask only affects PSOs that declare view instance masking by specifying the D3D12_VIEW_INSTANCING_FLAG_ENABLE_VIEW_INSTANCE_MASKING flag during their creation. Attempting to create a PSO that declares view instance masking will fail on adapters that don't support view instancing.

The view instance mask defaults to 0 which disables all views. This forces applications that declare view instance masking to explicitly choose the views to enable, otherwise nothing will be rendered. If the view instance mask enabled all views by default the application might not remember to disable unused views, resulting in lost performance due to wasted work.

Bundles don't inherit their view instance mask from their caller, defaulting to 0 instead. This is because the mask setting must be known when the bundle is recorded if it affects how an implementation records draws. The view instance mask set by a bundle does persist to the caller after the bundle completes, however. These inheritance semantics are similar to those of PSOs.

No shader code paths that are dependent on SV_ViewID are executed at any shader stage for view instances that are masked off and no clipping, viewport processing, or rasterization is performed. Implementations that inspect the mask during rendering can incur a small performance penalty over PSOs that don't declare view instance masking at all, but usually the penalty can be overcome by the performance savings that result from skipping the work associated with the masked off views. Depending on the frequency and amount of skipped work, the performance gains can be significant.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList2 interface (d3d12.h)

Article02/22/2024

Encapsulates a list of graphics commands for rendering, extending the interface to support writing immediate values directly to a buffer.

Note This interface was introduced in the Windows 10 Fall Creators Update, and as such is the latest version of the [ID3D12GraphicsCommandList](#) interface. Applications targeting the Windows 10 Fall Creators Update and later should use this interface instead of [ID3D12GraphicsCommandList1](#) or [ID3D12GraphicsCommandList](#).

Inheritance

The [ID3D12GraphicsCommandList2](#) interface inherits from [ID3D12GraphicsCommandList1](#). [ID3D12GraphicsCommandList2](#) also has these types of members:

Methods

The [ID3D12GraphicsCommandList2](#) interface has these methods.

[+] Expand table

[ID3D12GraphicsCommandList2::WriteBufferImmediate](#)

Writes a number of 32-bit immediate values to the specified buffer locations directly from the command stream. ([ID3D12GraphicsCommandList2::WriteBufferImmediate](#))

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12GraphicsCommandList](#)

[ID3D12GraphicsCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList2::WriteBufferImmediate method (d3d12.h)

Article 02/22/2024

Writes a number of 32-bit immediate values to the specified buffer locations directly from the command stream.

Syntax

C++

```
void WriteBufferImmediate(
    UINT                                     Count,
    [in]          const D3D12_WRITEBUFFERIMMEDIATE_PARAMETER *pParams,
    [in, optional] const D3D12_WRITEBUFFERIMMEDIATE_MODE     *pModes
);
```

Parameters

Count

The number of [D3D12_WRITEBUFFERIMMEDIATE_PARAMETER](#) structures that are pointed to by *pParams* and *pModes*.

[in] pParams

The address of an array containing a number of [D3D12_WRITEBUFFERIMMEDIATE_PARAMETER](#) structures equal to *Count*.

[in, optional] pModes

The address of an array containing a number of [D3D12_WRITEBUFFERIMMEDIATE_MODE](#) structures equal to *Count*. The default value is `null`; passing `null` causes the system to write all immediate values using [D3D12_WRITEBUFFERIMMEDIATE_MODE_DEFAULT](#).

Return value

None

Remarks

WriteBufferImmediate performs *Count* number of 32-bit writes: one for each value and destination specified in *pParams*.

The receiving buffer (resource) must be in the `D3D12_RESOURCE_STATE_COPY_DEST` state to be a valid destination for **WriteBufferImmediate**.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList3 interface (d3d12.h)

Article 02/22/2024

Encapsulates a list of graphics commands for rendering.

Inheritance

The **ID3D12GraphicsCommandList3** interface inherits from [ID3D12GraphicsCommandList2](#). **ID3D12GraphicsCommandList3** also has these types of members:

Methods

The **ID3D12GraphicsCommandList3** interface has these methods.

[+] Expand table

<p>ID3D12GraphicsCommandList3::SetProtectedResourceSession</p> <p>Specifies whether or not protected resources can be accessed by subsequent commands in the command list.</p>

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList3::SetProtectedResourceSession method (d3d12.h)

Article 02/22/2024

Specifies whether or not protected resources can be accessed by subsequent commands in the command list. By default, no protected resources are enabled. After calling **SetProtectedResourceSession** with a valid session, protected resources of the same type can refer to that session. After calling **SetProtectedResourceSession** with **NULL**, no protected resources can be accessed.

Syntax

C++

```
void SetProtectedResourceSession(  
    [in, optional] ID3D12ProtectedResourceSession *pProtectedResourceSession  
) ;
```

Parameters

[in, optional] pProtectedResourceSession

Type: [ID3D12ProtectedResourceSession*](#)

An optional pointer to an **ID3D12ProtectedResourceSession**. You can obtain an **ID3D12ProtectedResourceSession** by calling [ID3D12Device4::CreateProtectedResourceSession](#).

Return value

If set, indicates that protected resources can be accessed with the given session. Access to protected resources can only happen after **SetProtectedResourceSession** is called with a valid session. The command list state is cleared when calling this method. If you pass **NULL**, then no protected resources can be accessed.

Requirements

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList3](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4 interface (d3d12.h)

Article 07/22/2021

Encapsulates a list of graphics commands for rendering, extending the interface to support ray tracing and render passes.

Inheritance

The **ID3D12GraphicsCommandList4** interface inherits from [ID3D12GraphicsCommandList3](#). **ID3D12GraphicsCommandList4** also has these types of members:

Methods

The **ID3D12GraphicsCommandList4** interface has these methods.

[+] Expand table

ID3D12GraphicsCommandList4::BeginRenderPass
Marks the beginning of a render pass by binding a set of output resources for the duration of the render pass. These bindings are to one or more render target views (RTVs), and/or to a depth stencil view (DSV).
ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure
Performs a raytracing acceleration structure build on the GPU and optionally outputs post-build information immediately after the build.
ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure
Copies a source acceleration structure to destination memory while applying the specified transformation.
ID3D12GraphicsCommandList4::DispatchRays
Launch the threads of a ray generation shader.
ID3D12GraphicsCommandList4::EmitRaytracingAccelerationStructurePostbuildInfo
Emits post-build properties for a set of acceleration structures. This enables applications to know

the output resource requirements for performing acceleration structure operations via ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure.
ID3D12GraphicsCommandList4::EndRenderPass
Marks the ending of a render pass.
ID3D12GraphicsCommandList4::ExecuteMetaCommand
Records the execution (or invocation) of the specified meta command into a graphics command list.
ID3D12GraphicsCommandList4::InitializeMetaCommand
Initializes the specified meta command.
ID3D12GraphicsCommandList4::SetPipelineState1
Sets a state object on the command list.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList3](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList4::BeginRenderPass method (d3d12.h)

Article 02/22/2024

Marks the beginning of a render pass by binding a set of output resources for the duration of the render pass. These bindings are to one or more render target views (RTVs), and/or to a depth stencil view (DSV).

Syntax

C++

```
void BeginRenderPass(  
    UINT NumRenderTargets,  
    const D3D12_RENDER_PASS_RENDER_TARGET_DESC *pRenderTarget,  
    const D3D12_RENDER_PASS_DEPTH_STENCIL_DESC *pDepthStencil,  
    D3D12_RENDER_PASS_FLAGS Flags  
)
```

Parameters

NumRenderTargets

A **UINT**. The number of render targets being bound.

pRenderTarget

A pointer to a constant [D3D12_RENDER_PASS_RENDER_TARGET_DESC](#), which describes bindings (fixed for the duration of the render pass) to one or more render target views (RTVs), as well as their beginning and ending access characteristics.

pDepthStencil

A pointer to a constant [D3D12_RENDER_PASS_DEPTH_STENCIL_DESC](#), which describes a binding (fixed for the duration of the render pass) to a depth stencil view (DSV), as well as its beginning and ending access characteristics.

Flags

A [D3D12_RENDER_PASS_FLAGS](#). The nature/requirements of the render pass; for example, whether it is a suspending or a resuming render pass, or whether it wants to

write to unordered access view(s).

Return value

None

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[EndRenderPass](#)

[ID3D12GraphicsCommandList4](#)

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure method (d3d12.h)

Article 02/22/2024

Performs a raytracing acceleration structure build on the GPU and optionally outputs post-build information immediately after the build.

Syntax

C++

```
void BuildRaytracingAccelerationStructure(
    [in] const D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC
    *pDesc,
    [in] UINT
    NumPostbuildInfoDescs,
    [in] const D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC
    *pPostbuildInfoDescs
);
```

Parameters

[in] pDesc

Description of the acceleration structure to build.

[in] NumPostbuildInfoDescs

Size of the *pPostbuildInfoDescs* array. Set to 0 if no post-build info is needed.

[in] pPostbuildInfoDescs

Optional array of descriptions for post-build info to generate describing properties of the acceleration structure that was built.

Return value

None

Remarks

This method can be called on graphics or compute command lists but not from bundles.

Post-build information can also be obtained separately from an already built acceleration structure by calling [EmitRaytracingAccelerationStructurePostbuildInfo](#). The advantage of generating post-build info along with a build is that a barrier isn't needed in between the build completing and requesting post-build information, enabling scenarios where the app needs the post-build info right away.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure method (d3d12.h)

Article 10/13/2021

Copies a source acceleration structure to destination memory while applying the specified transformation.

Syntax

C++

```
void CopyRaytracingAccelerationStructure(
    [in] D3D12_GPU_VIRTUAL_ADDRESS
DestAccelerationStructureData,
    [in] D3D12_GPU_VIRTUAL_ADDRESS
SourceAccelerationStructureData,
    [in] D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE Mode
);
```

Parameters

[in] DestAccelerationStructureData

The destination memory. The required size can be discovered by calling [EmitRaytracingAccelerationStructurePostbuildInfo](#) beforehand, if necessary for the specified *Mode*.

The destination start address must be aligned to 256 bytes, defined as [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT](#), regardless of the specified *Mode*.

The destination memory range cannot overlap source. Otherwise, results are undefined.

The resource state that the memory pointed to must be in depends on the *Mode* parameter. For more information, see [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE](#).

[in] SourceAccelerationStructureData

The address of the acceleration structure or other type of data to copy/transform based on the specified *Mode*. The data remains unchanged and usable. The operation only copies the data pointed to by *SourceAccelerationStructureData* and not any other data, such as acceleration structures, that the source data may point to. For example, in the case of a top-level acceleration structure, any bottom-level acceleration structures that it points to are not copied in the operation.

The source memory must be aligned to 256 bytes, defined as [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT](#), regardless of the specified *Mode*.

The resource state that the memory pointed to must be in depends on the *Mode* parameter. For more information, see [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE](#).

[in] *Mode*

The type of copy operation to perform. For more information, see [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE](#).

Return value

None

Remarks

Since raytracing acceleration structures may contain internal pointers and have a device dependent opaque layout, copying them around or otherwise manipulating them requires a dedicated API so that drivers can handle the requested operation.

This method can be called from graphics or compute command lists but not from bundles.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::DispatchRays method (d3d12.h)

Article 02/22/2024

Launch the threads of a ray generation shader.

Syntax

C++

```
void DispatchRays(  
    [in] const D3D12_DISPATCH_RAYS_DESC *pDesc  
);
```

Parameters

[in] pDesc

A description of the ray dispatch

Return value

None

Remarks

This method can be called from graphics or compute command lists and bundles.

A raytracing pipeline state must be set on the command list. Otherwise, the behavior of this call is undefined.

There are 3 dimensions passed in to set the grid size: width/height/depth. These dimensions are constrained such that width * height * depth <= 2^30. Exceeding this produces undefined behavior. If any grid dimension is 0, no threads are launched.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::EmitRaytracingAccelerationStructurePostbuildInfo method (d3d12.h)

Article 07/13/2022

Emits post-build properties for a set of acceleration structures. This enables applications to know the output resource requirements for performing acceleration structure operations via [ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure](#).

Syntax

C++

```
void EmitRaytracingAccelerationStructurePostbuildInfo(
    [in] const D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC
*pDesc,
    [in] UINT
NumSourceAccelerationStructures,
    [in] const D3D12_GPU_VIRTUAL_ADDRESS
*pSourceAccelerationStructureData
);
```

Parameters

[in] *pDesc*

A [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC](#) object describing post-build information to generate.

[in] *NumSourceAccelerationStructures*

Number of pointers to acceleration structure GPU virtual addresses pointed to by *pSourceAccelerationStructureData*. This number also affects the destination (output), which will be a contiguous array of **NumSourceAccelerationStructures** output structures, where the type of the structures depends on *InfoType* field of the supplied in the *pDesc* description.

[in] *pSourceAccelerationStructureData*

Pointer to array of GPU virtual addresses of size *NumSourceAccelerationStructures*.

The address must be aligned to 256 bytes, defined as [D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BYTE_ALIGNMENT](#).

The memory pointed to must be in state [D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE](#).

Return value

None

Remarks

This method can be called from graphics or compute command lists but not from bundles.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12GraphicsCommandList4::EndRenderPass method (d3d12.h)

Article 02/22/2024

Marks the ending of a render pass.

Syntax

C++

```
void EndRenderPass();
```

Return value

None

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[BeginRenderPass](#)

[ID3D12GraphicsCommandList4](#)

[Rendering](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::ExecuteMetaCommand method (d3d12.h)

Article 10/13/2021

Records the execution (or invocation) of the specified meta command into a graphics command list.

Call [ID3D12GraphicsCommandList4::InitializeMetaCommand](#) before executing a meta command. During invocation, you can specify overrides for values of any of the runtime parameters. You can execute multiple meta commands on the same graphics command list. And you can execute the same meta command multiple times on the same command list.

With a PIX capture taken with the use of meta commands, you can play that back on the same hardware configuration. But, by design, it's not portable to other GPUs.

Syntax

C++

```
void ExecuteMetaCommand(  
    [in]          ID3D12MetaCommand *pMetaCommand,  
    [in, optional] const void     *pExecutionParametersData,  
    [in]          SIZE_T          ExecutionParametersDataSizeInBytes  
)
```

Parameters

[in] pMetaCommand

A pointer to an [ID3D12MetaCommand](#) representing the meta command to initialize.

[in, optional] pExecutionParametersData

An optional pointer to a constant structure containing the values of the parameters for executing the meta command.

[in] ExecutionParametersDataSizeInBytes

A [SIZE_T](#) containing the size of the structure pointed to by *pExecutionParametersData*, if set, otherwise 0.

Return value

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Remarks

Your application is responsible for setting up the resources supplied to a meta command in the state required according to the meta command specification. The meta command definition specification defines the expected resource state for each parameter. Your application is responsible for inserting unordered access view (UAV) barriers for input resources before the meta command's algorithm can consume them. You're also responsible for inserting the UAV barrier for the output resources when you intend to read them back.

During an algorithm invocation, the driver may insert as many UAV barriers to output resources as are needed to synchronize the output resource usage in the algorithm implementation. From your application's point of view, you should assume that all out and in/out resources are written to by the meta command, including scratch memory.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::InitializeMetaCommand method (d3d12.h)

Article 02/22/2024

Initializes the specified meta command.

You must initialize a meta command at least once prior (on the GPU's timeline) to executing it. Initializing gives the implementation the chance to perform any work necessary to accelerate the invocation of the meta command. You must supply the sufficient resource parameters, including the persistent cache resource.

Syntax

C++

```
void InitializeMetaCommand(
    [in]          ID3D12MetaCommand *pMetaCommand,
    [in, optional] const void      *pInitializationParametersData,
    [in]          SIZE_T           InitializationParametersDataSizeInBytes
);
```

Parameters

[in] pMetaCommand

A pointer to an [ID3D12MetaCommand](#) representing the meta command to initialize.

[in, optional] pInitializationParametersData

An optional pointer to a constant structure containing the values of the parameters for initializing the meta command.

[in] InitializationParametersDataSizeInBytes

A [SIZE_T](#) containing the size of the structure pointed to by *pInitializationParametersData*, if set, otherwise 0.

Return value

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList4::SetPipelineState1 method (d3d12.h)

Article 02/22/2024

Sets a state object on the command list.

Syntax

C++

```
void SetPipelineState1(  
    ID3D12StateObject *pStateObject  
);
```

Parameters

pStateObject

The state object to set on the command list. In the current release, this can only be of type [D3D12_STATE_OBJECT_TYPE_RAYTRACING_PIPELINE](#).

Return value

None

Remarks

This method can be called from graphics or compute command lists and bundles.

This method is an alternative to [ID3D12GraphicsCommandList::SetPipelineState](#), which is only defined for graphics and compute shaders. There is only one pipeline state active on a command list at a time, so either call sets the current pipeline state. The distinction between the calls is that each sets particular types of pipeline state only. In the current release, `SetPipelineState1` is only used for setting raytracing pipeline state.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1809 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList5 interface (d3d12.h)

Article 02/22/2024

Encapsulates a list of graphics commands for rendering, extending the interface to support variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Inheritance

The **ID3D12GraphicsCommandList5** interface inherits from the [ID3D12GraphicsCommandList4](#) interface.

Inheritance

The **ID3D12GraphicsCommandList5** interface inherits from the [ID3D12GraphicsCommandList4](#) interface.

Methods

The **ID3D12GraphicsCommandList5** interface has these methods.

[+] [Expand table](#)

<p>ID3D12GraphicsCommandList5::RSSetShadingRate</p> <p>The ID3D12GraphicsCommandList5::RSSetShadingRate method (d3d12.h) sets the base shading rate, and combiners, for variable-rate shading (VRS).</p>
<p>ID3D12GraphicsCommandList5::RSSetShadingRateImage</p> <p>The ID3D12GraphicsCommandList5::RSSetShadingRateImage method (d3d12.h) sets the screen-space shading-rate image for variable-rate shading (VRS).</p>

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList5::RSSetShadingRate method (d3d12.h)

Article 08/19/2022

Sets the base shading rate, and combiners, for variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

C++

```
void RSSetShadingRate(  
    D3D12_SHADING_RATE           baseShadingRate,  
    const D3D12_SHADING_RATE_COMBINER *combiners  
) ;
```

Parameters

`baseShadingRate`

Type: [D3D12_SHADING_RATE](#)

A constant from the [D3D12_SHADING_RATE](#) enumeration describing the base shading rate to set.

`combiners`

Type: [const D3D12_SHADING_RATE_COMBINER*](#)

An optional pointer to a constant array of [D3D12_SHADING_RATE_COMBINER](#) containing the shading rate combiners to set. The count of [D3D12_SHADING_RATE_COMBINER](#) elements in the array must be equal to the constant [D3D12_RS_SET_SHADING_RATE_COMBINER_COUNT](#), which is equal to 2.

Because per-primitive and screen-space image-based VRS isn't supported on Tier1 [Variable-rate shading \(VRS\)](#), for these values to be meaningful, the adapter requires Tier2 VRS support. See [D3D12_FEATURE_DATA_D3D12_OPTIONS6](#) and [D3D12_VARIABLE_SHADING_RATE_TIER](#).

A NULL pointer is equivalent to the default shading combiners, which are both [D3D12_SHADING_RATE_COMBINER_PASSTHROUGH](#).

The algorithm for final shading-rate is determined by the following.

C++

```
postRasterizerRate = ApplyCombiner(Combiners[0], CommandListShadingRate,  
Primitive->PrimitiveSpecifiedShadingRate);  
finalRate = ApplyCombiner(Combiners[1], postRasterizerRate,  
ScreenSpaceImage[xy]);
```

where `ApplyCombiner` is

C++

```
UINT ApplyCombiner(D3D12_SHADING_RATE_COMBINER combiner, UINT a, UINT b)  
{  
    MaxShadingRate = options6.AdditionalShadingRatesSupported ? 4 : 2;  
    switch (combiner)  
    {  
        case D3D12_SHADING_RATE_COMBINER_PASSTHROUGH: // default  
            return a;  
        case D3D12_SHADING_RATE_COMBINER_OVERRIDE:  
            return b;  
        case D3D12_SHADING_RATE_COMBINER_MAX:  
            return max(a, b);  
        case D3D12_SHADING_RATE_COMBINER_MIN:  
            return min(a, b);  
        case D3D12_SHADING_RATE_COMBINER_SUM:  
            return min(MaxShadingRate, a + b);  
        case default:  
            return a;  
    }  
}
```

Return value

None

Requirements

[\[\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList5::RSSetShadingRateImage method (d3d12.h)

Article 08/19/2022

Sets the screen-space shading-rate image for variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#). This method requires Tier2 [Variable-rate shading \(VRS\)](#) support. See [D3D12_FEATURE_DATA_D3D12_OPTIONS6](#) and [D3D12_VARIABLE_SHADING_RATE_TIER](#).

Syntax

C++

```
void RSSetShadingRateImage(  
    ID3D12Resource *shadingRateImage  
)
```

Parameters

shadingRateImage

Type: [ID3D12Resource*](#)

An optional pointer to an [ID3D12Resource](#) representing a screen-space shading-rate image. If **NULL**, the effect is the same as having a shading-rate image where all values are a shading rate of 1x1.

This texture must have the [D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE](#) state applied.

The tile-size of the shading-rate image can be determined via [D3D12_FEATURE_DATA_D3D12_OPTIONS6](#). The size of the shading-rate image should therefore be

```
ceil((float)rtWidth / VRSTileSize), ceil((float)rtHeight / VRSTileSize)
```

The shading-rate image must be a 2D texture with a single mip, and format [DXGI_FORMAT_R8_UINT](#). Each texel must be a value corresponding to

[D3D12_SHADING_RATE](#). It must have layout [D3D12_TEXTURE_LAYOUT_UNKNOWN](#) and can't be a depth-stencil, render-target, simultaneous-access, or cross-adapter resource.

As (0, 0) is the top left in DirectX, a too-small or large shading-rate image results in the bottom or right having no shading-rate image area, or the image extending in these directions. When there is excess, it is ignored (but legal), and when the image is too small, all out-of-bounds areas in the bottom and right will have the default shading rate of 1x1 from the image (however, this does not mean that is the final shading rate. The combiners will still be applied to this 1x1 default value).

Return value

None

Remarks

For the screen-space shading-rate image to take affect,

[ID3D12GraphicsCommandList5::RSSetShadingRate](#) must have been called to set the combiners for shading. Else, with the default combiners (both [D3D12_SHADING_RATE_COMBINER_PASSTHROUGH](#)), the screen-space shading-rate image is ignored in determining shading granularity.

The second combiner passed to [\[ID3D12GraphicsCommandList5::RSSetShadingRate\]](#) is the one which applies to the shading-rate image, which occurs after the global shading rate and the per-primitive shading rate have been combined.

The algorithm for final shading-rate is determined by

C++

```
postRasterizerRate = ApplyCombiner(Combiners[0], CommandListShadingRate,
Primitive->PrimitiveSpecifiedShadingRate);
finalRate = ApplyCombiner(Combiners[1], postRasterizerRate,
ScreenSpaceImage[xy]);
```

where `ApplyCombiner` is

C++

```
UINT ApplyCombiner(D3D12_SHADING_RATE_COMBINER combiner, UINT a, UINT b)
{
    MaxShadingRate = options6.AdditionalShadingRatesSupported ? 4 : 2;
    switch (combiner)
    {
```

```
        case D3D12_SHADING_RATE_COMBINER_PASSTHROUGH: // default
            return a;
        case D3D12_SHADING_RATE_COMBINER_OVERRIDE:
            return b;
        case D3D12_SHADING_RATE_COMBINER_MAX:
            return max(a, b);
        case D3D12_SHADING_RATE_COMBINER_MIN:
            return min(a, b);
        case D3D12_SHADING_RATE_COMBINER_SUM:
            return min(MaxShadingRate, a + b);
        case default:
            return a;
    }
}
```

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h

See also

[Variable-rate shading \(VRS\)](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12GraphicsCommandList6 interface (d3d12.h)

Article02/22/2024

Inheritance

The **ID3D12GraphicsCommandList6** interface inherits from the **ID3D12GraphicsCommandList5** interface.

Methods

The **ID3D12GraphicsCommandList6** interface has these methods.

[+] Expand table

ID3D12GraphicsCommandList6::DispatchMesh

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList6::DispatchMesh method (d3d12.h)

Article02/22/2024

Syntax

C++

```
void DispatchMesh(  
    UINT ThreadGroupCountX,  
    UINT ThreadGroupCountY,  
    UINT ThreadGroupCountZ  
);
```

Parameters

ThreadGroupCountX

ThreadGroupCountY

ThreadGroupCountZ

Return value

None

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12GraphicsCommandList7 interface (d3d12.h)

Article 02/22/2024

TBD

Requires the DirectX 12 Agility SDK 1.7 or later.

Inheritance

The **ID3D12GraphicsCommandList7** interface inherits from the **ID3D12GraphicsCommandList6** interface.

Methods

The **ID3D12GraphicsCommandList7** interface has these methods.

[+] Expand table

<p>ID3D12GraphicsCommandList7::Barrier</p> <p>Adds a collection of barriers into a graphics command list recording.</p>

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

ID3D12GraphicsCommandList7::Barrier method (d3d12.h)

Article 02/22/2024

Adds a collection of barriers into a graphics command list recording.

Requires the DirectX 12 Agility SDK 1.608 or later.

Syntax

C++

```
void Barrier(
    UINT32 NumBarrierGroups,
    const D3D12_BARRIER_GROUP *pBarrierGroups
);
```

Parameters

NumBarrierGroups

Number of barrier groups pointed to by *pBarrierGroups*.

pBarrierGroups

Pointer to an array of [D3D12_BARRIER_GROUP](#) objects.

Return value

None

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Requirement	Value
Library	D3d12.lib
DLL	D3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Heap interface (d3d12.h)

Article02/22/2024

A heap is an abstraction of contiguous memory allocation, used to manage physical memory. This heap can be used with [ID3D12Resource](#) objects to support placed resources or reserved resources.

Inheritance

The **ID3D12Heap** interface inherits from [ID3D12Pageable](#). **ID3D12Heap** also has these types of members:

Methods

The **ID3D12Heap** interface has these methods.

[] [Expand table](#)

ID3D12Heap::GetDesc	
Gets the heap description.	

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Heap::GetDesc method (d3d12.h)

Article02/22/2024

Gets the heap description.

Syntax

C++

```
D3D12_HEAP_DESC GetDesc();
```

Return value

Type: [D3D12_HEAP_DESC](#)

Returns the [D3D12_HEAP_DESC](#) structure that describes the heap.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Heap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Heap1 interface (d3d12.h)

Article02/22/2024

Inheritance

The ID3D12Heap1 interface inherits from the ID3D12Heap interface.

Methods

The ID3D12Heap1 interface has these methods.

 Expand table

ID3D12Heap1::GetProtectedResourceSession

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Heap1::GetProtectedResourceSession method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetProtectedResourceSession(  
    REFIID riid,  
    void    **ppProtectedSession  
>;
```

Parameters

riid

ppProtectedSession

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12LifetimeOwner interface (d3d12.h)

Article 02/22/2024

Represents an application-defined callback used for being notified of lifetime changes of an object.

Inheritance

The ID3D12LifetimeOwner interface inherits from the IUnknown interface.

Methods

The ID3D12LifetimeOwner interface has these methods.

[+] Expand table

<p>ID3D12LifetimeOwner::LifetimeStateUpdated</p> <p>Called when the lifetime state of a lifetime-tracked object changes.</p>

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12LifetimeOwner::LifetimeStateUpdated method (d3d12.h)

Article02/22/2024

Called when the lifetime state of a lifetime-tracked object changes.

Syntax

C++

```
void LifetimeStateUpdated(  
    D3D12_LIFETIME_STATE NewState  
)
```

Parameters

NewState

Type: [D3D12_LIFETIME_STATE](#)

The new state.

Return value

None

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12LifetimeTracker interface (d3d12.h)

Article 02/22/2024

Represents facilities for controlling the lifetime a lifetime-tracked object.

Inheritance

The **ID3D12LifetimeTracker** interface inherits from the **ID3D12DeviceChild** interface.

Methods

The **ID3D12LifetimeTracker** interface has these methods.

[+] Expand table

<p>ID3D12LifetimeTracker::DestroyOwnedObject</p> <p>Destroys a lifetime-tracked object.</p>

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12LifetimeTracker::DestroyOwnedObject method (d3d12.h)

Article 02/22/2024

Destroys a lifetime-tracked object.

Syntax

C++

```
HRESULT DestroyOwnedObject(  
    [in] ID3D12DeviceChild *pObject  
);
```

Parameters

[in] pObject

Type: [ID3D12DeviceChild*](#)

A pointer to an [ID3D12DeviceChild](#) interface representing the lifetime-tracked object.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12MetaCommand interface (d3d12.h)

Article 02/22/2024

Represents a meta command. A meta command is a Direct3D 12 object representing an algorithm that is accelerated by independent hardware vendors (IHVs). It's an opaque reference to a command generator that is implemented by the driver.

The lifetime of a meta command is tied to the lifetime of the command list that references it. So, you should only free a meta command if no command list referencing it is currently executing on the GPU.

A meta command can encapsulate a set of pipeline state objects (PSOs), bindings, intermediate resource states, and Draw/Dispatch calls. You can think of the signature of a meta command as being similar to a C-style function, with multiple in/out parameters, and no return value.

Inheritance

The [ID3D12MetaCommand](#) interface inherits from [ID3D12Pageable](#).

[ID3D12MetaCommand](#) also has these types of members:

Methods

The [ID3D12MetaCommand](#) interface has these methods.

[] [Expand table](#)

[ID3D12MetaCommand::GetRequiredParameterResourceSize](#)

Retrieves the amount of memory required for the specified runtime parameter resource for a meta command, for the specified stage.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12MetaCommand::GetRequiredParameterResourceSize method (d3d12.h)

Article 02/22/2024

Retrieves the amount of memory required for the specified runtime parameter resource for a meta command, for the specified stage.

Syntax

C++

```
UINT64 GetRequiredParameterResourceSize(  
    [in] D3D12_META_COMMAND_PARAMETER_STAGE Stage,  
    [in] UINT ParameterIndex  
) ;
```

Parameters

[in] Stage

Type: D3D12_META_COMMAND_PARAMETER_STAGE

A D3D12_META_COMMAND_PARAMETER_STAGE specifying the stage to which the parameter belongs.

[in] ParameterIndex

Type: [UINT](#)

The zero-based index of the parameter within the stage.

Return value

Type: [UINT64](#)

The number of bytes required for the specified runtime parameter resource.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12MetaCommand](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Object interface (d3d12.h)

Article02/22/2024

An interface from which [ID3D12Device](#) and [ID3D12DeviceChild](#) inherit from. It provides methods to associate private data and annotate object names.

Inheritance

The [ID3D12Object](#) interface inherits from the [IUnknown](#) interface. [ID3D12Object](#) also has these types of members:

Methods

The [ID3D12Object](#) interface has these methods.

[+] Expand table

ID3D12Object::GetPrivateData
Gets application-defined data from a device object.
ID3D12Object::SetName
Associates a name with the device object. This name is for use in debug diagnostics and tools.
ID3D12Object::SetPrivateData
Sets application-defined data to a device object and associates that data with an application-defined GUID.
ID3D12Object::SetPrivateDataInterface
Associates an IUnknown-derived interface with the device object, and associates that interface with an application-defined GUID.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Object::GetPrivateData method (d3d12.h)

Article 02/22/2024

Gets application-defined data from a device object.

Syntax

C++

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

Parameters

[in] guid

Type: [REFGUID](#)

The GUID that is associated with the data.

[in, out] pDataSize

Type: [UINT*](#)

A pointer to a variable that on input contains the size, in bytes, of the buffer that *pData* points to, and on output contains the size, in bytes, of the amount of data that *GetPrivateData* retrieved.

[out, optional] pData

Type: [void*](#)

A pointer to a memory block that receives the data from the device object if *pDataSize* points to a value that specifies a buffer large enough to hold the data.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

If the data returned is a pointer to an [IUnknown](#), or one of its derivative classes, which was previously set by SetPrivateDataInterface, that interface will have its reference count incremented before the private data is returned.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Object](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Object::SetName method (d3d12.h)

Article 02/22/2024

Associates a name with the device object. This name is for use in debug diagnostics and tools.

Syntax

C++

```
HRESULT SetName(  
    [in] LPCWSTR Name  
);
```

Parameters

[in] Name

Type: **LPCWSTR**

A NULL-terminated **UNICODE** string that contains the name to associate with the device object.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method takes **UNICODE** names.

Note that this is simply a convenience wrapper around [ID3D12Object::SetPrivateData](#) with **WKPID_D3DDescribeObjectNameW**. Therefore names which are set with [SetName](#) can be retrieved with [ID3D12Object::GetPrivateData](#) with the same GUID. Additionally, D3D12 supports narrow strings for names, using the **WKPID_D3DDescribeObjectName** GUID directly instead.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Direct3D 12 Programming Environment Setup](#)

[ID3D12Object](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Object::SetPrivateData method (d3d12.h)

Article 02/22/2024

Sets application-defined data to a device object and associates that data with an application-defined **GUID**.

Syntax

C++

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

Parameters

[in] `guid`

Type: [REFGUID](#)

The **GUID** to associate with the data.

[in] `DataSize`

Type: [UINT](#)

The size in bytes of the data.

[in, optional] `pData`

Type: `const void*`

A pointer to a memory block that contains the data to be stored with this device object. If *pData* is **NULL**, *DataSize* must also be 0, and any data that was previously associated with the **GUID** specified in *guid* will be destroyed.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Rather than using the Direct3D 11 debug object naming scheme of calling `ID3D12Object::SetPrivateData` using `WKPID_D3DDescribeObjectName` with an ASCII name, call `ID3D12Object::SetName` with a UNICODE name.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Object](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Object::SetPrivateDataInterface method (d3d12.h)

Article 02/22/2024

Associates an [IUnknown](#)-derived interface with the device object, and associates that interface with an application-defined GUID.

Syntax

C++

```
HRESULT SetPrivateDataInterface(  
    [in]          REFGUID      guid,  
    [in, optional] const IUnknown *pData  
>;
```

Parameters

[in] guid

Type: [REFGUID](#)

The GUID to associate with the interface.

[in, optional] pData

Type: [const IUnknown*](#)

A pointer to the [IUnknown](#)-derived interface to be associated with the device object. Its reference count is incremented when set, and its reference count is decremented when either the [ID3D12Object](#) is destroyed, or when the data is overwritten by calling [SetPrivateData](#) or [SetPrivateDataInterface](#) with the same GUID.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 return codes](#).

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12Object](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12Pageable interface (d3d12.h)

Article 02/22/2024

An interface from which many other core interfaces inherit from. It indicates that the object type encapsulates some amount of GPU-accessible memory; but does not strongly indicate whether the application can manipulate the object's residency.

Inheritance

The **ID3D12Pageable** interface inherits from the **ID3D12DeviceChild** interface.

Remarks

For more details, refer to [Memory Management in Direct3D 12](#) and the [MakeResident](#) method reference.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptor Heaps](#)

[ID3D12DeviceChild](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12PipelineLibrary interface (d3d12.h)

Article 02/22/2024

Manages a pipeline library, in particular loading and retrieving individual PSOs.

Inheritance

The **ID3D12PipelineLibrary** interface inherits from [ID3D12DeviceChild](#).

ID3D12PipelineLibrary also has these types of members:

Methods

The **ID3D12PipelineLibrary** interface has these methods.

 [Expand table](#)

ID3D12PipelineLibrary::GetSerializedSize
Returns the amount of memory required to serialize the current contents of the database.
ID3D12PipelineLibrary::LoadComputePipeline
Retrieves the requested PSO from the library. The input desc is matched against the data in the current library database, and remembered in order to prevent duplication of PSO contents.
ID3D12PipelineLibrary::LoadGraphicsPipeline
Retrieves the requested PSO from the library.
ID3D12PipelineLibrary::Serialize
Writes the contents of the library to the provided memory, to be provided back to the runtime at a later time.
ID3D12PipelineLibrary::StorePipeline
Adds the input PSO to an internal database with the corresponding name.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12DeviceChild](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineLibrary::GetSerializedSize method (d3d12.h)

Article02/22/2024

Returns the amount of memory required to serialize the current contents of the database.

Syntax

C++

```
SIZE_T GetSerializedSize();
```

Return value

Type: SIZE_T

This method returns a SIZE_T object, containing the size required in bytes.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineLibrary::LoadComputePipeline method (d3d12.h)

Article 02/22/2024

Retrieves the requested PSO from the library. The input desc is matched against the data in the current library database, and remembered in order to prevent duplication of PSO contents.

Syntax

C++

```
HRESULT LoadComputePipeline(
    [in]    LPCWSTR                                pName,
    [in]    const D3D12_COMPUTE_PIPELINE_STATE_DESC *pDesc,
    REFIID                                         riid,
    [out]   void*                                     **ppPipelineState
);
```

Parameters

[in] pName

Type: **LPCWSTR**

The unique name of the PSO.

[in] pDesc

Type: **const D3D12_COMPUTE_PIPELINE_STATE_DESC***

Specifies a description of the required PSO in a **D3D12_COMPUTE_PIPELINE_STATE_DESC** structure. This input description is matched against the data in the current library database, and stored in order to prevent duplication of PSO contents.

riid

Type: **REFIID**

Specifies a REFIID for the **ID3D12PipelineState** object. Typically set this, and the following parameter, with the macro **IID_PPV_ARGS(&PSO1)**, where *PSO1* is the name of the object.

[out] ppPipelineState

Type: **void****

Specifies a pointer that will reference the returned PSO.

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code, which can include E_INVALIDARG if the name doesn't exist, or if the input description doesn't match the data in the library, and E_OUTOFMEMORY if unable to allocate the return PSO.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12PipelineLibrary::LoadGraphicsPipeline method (d3d12.h)

Article 02/22/2024

Retrieves the requested PSO from the library.

Syntax

C++

```
HRESULT LoadGraphicsPipeline(
    [in]    LPCWSTR                      pName,
    [in]    const D3D12_GRAPHICS_PIPELINE_STATE_DESC *pDesc,
    REFIID                         riid,
    [out]   void                          **ppPipelineState
);
```

Parameters

[in] pName

Type: **LPCWSTR**

The unique name of the PSO.

[in] pDesc

Type: **const D3D12_GRAPHICS_PIPELINE_STATE_DESC***

Specifies a description of the required PSO in a **D3D12_GRAPHICS_PIPELINE_STATE_DESC** structure. This input description is matched against the data in the current library database, and stored in order to prevent duplication of PSO contents.

riid

Type: **REFIID**

Specifies a REFIID for the **ID3D12PipelineState** object. Typically set this, and the following parameter, with the macro **IID_PPV_ARGS(&PSO1)**, where *PSO1* is the name of the object.

[out] ppPipelineState

Type: **void****

Specifies a pointer that will reference the returned PSO.

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code, which can include E_INVALIDARG if the name doesn't exist, or if the input description doesn't match the data in the library, and E_OUTOFMEMORY if unable to allocate the return PSO.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12PipelineLibrary::Serialize method (d3d12.h)

Article 02/22/2024

Writes the contents of the library to the provided memory, to be provided back to the runtime at a later time.

Syntax

C++

```
HRESULT Serialize(
    [out] void     *pData,
    SIZE_T DataSizeInBytes
);
```

Parameters

[out] pData

Type: **void***

Specifies a pointer to the data. This memory must be readable and writable up to the input size. This data can be saved and provided to [CreatePipelineLibrary](#) at a later time, including future instances of this or other processes. The data becomes invalidated if the runtime or driver is updated, and is not portable to other hardware or devices.

ContentSizeInBytes

Type: **SIZE_T**

The size provided must be at least the size returned from [GetSerializedSize](#).

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code, including E_INVALIDARG if the buffer provided isn't big enough.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineLibrary::StorePipeline method (d3d12.h)

Article 02/22/2024

Adds the input PSO to an internal database with the corresponding name.

Syntax

C++

```
HRESULT StorePipeline(
    [in, optional] LPCWSTR pName,
    [in]           ID3D12PipelineState *pPipeline
);
```

Parameters

[in, optional] pName

Type: **LPCWSTR**

Specifies a unique name for the library. Overwriting is not supported.

[in] pPipeline

Type: **ID3D12PipelineState***

Specifies the **ID3D12PipelineState** to add.

Return value

Type: **HRESULT**

This method returns an HRESULT success or error code, including E_INVALIDARG if the name already exists, E_OUTOFMEMORY if unable to allocate storage in the library.

Remarks

Refer to the remarks and examples for [CreatePipelineLibrary](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12PipelineLibrary1 interface (d3d12.h)

Article 02/22/2024

Manages a pipeline library. This interface extends [ID3D12PipelineLibrary](#) to load PSOs from a pipeline state stream description.

Inheritance

The [ID3D12PipelineLibrary1](#) interface inherits from [ID3D12PipelineLibrary](#).

[ID3D12PipelineLibrary1](#) also has these types of members:

Methods

The [ID3D12PipelineLibrary1](#) interface has these methods.

[] [Expand table](#)

<p>ID3D12PipelineLibrary1::LoadPipeline</p> <p>Retrieves the requested PSO from the library. The pipeline stream description is matched against the library database, and remembered in order to prevent duplication of PSO contents.</p>

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineLibrary1::LoadPipeline method (d3d12.h)

Article 02/22/2024

Retrieves the requested PSO from the library. The pipeline stream description is matched against the library database, and remembered in order to prevent duplication of PSO contents.

Syntax

C++

```
HRESULT LoadPipeline(
    [in]    LPCWSTR                      pName,
    [in]    const D3D12_PIPELINE_STATE_STREAM_DESC *pDesc,
    REFIID                         riid,
    [out]   void*                         **ppPipelineState
);
```

Parameters

[in] pName

Type: **LPCWSTR**

SAL: *In*

The unique name of the PSO.

[in] pDesc

Type: **const D3D12_PIPELINE_STATE_STREAM_DESC***

SAL: *In*

Describes the required PSO using a **D3D12_PIPELINE_STATE_STREAM_DESC** structure. This description is matched against the library database, and stored in order to prevent duplication of PSO contents.

riid

Type: **REFIID**

Specifies a REFIID for the [ID3D12PipelineState](#) object.

Your app should typically set this argument and the following argument, *ppPipelineState*, by using the macro IID_PPV_ARGS(&PSO1), where PSO1 is the name of the object.

[out] *ppPipelineState*

Type: **void****

SAL: *COM_Outptr*

Specifies the pointer that will reference the PSO after the function successfully returns.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code, which can include E_INVALIDARG if the name doesn't exist or the stream description doesn't match the data in the library, and E_OUTOFMEMORY if the function is unable to allocate the resulting PSO.

Remarks

This function takes the pipeline description as a [D3D12_PIPELINE_STATE_STREAM_DESC](#) and is a replacement for the [ID3D12PipelineLibrary::LoadGraphicsPipeline](#) and [ID3D12PipelineLibrary::LoadComputePipeline](#) functions, which take their pipeline description as the less-flexible [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structs, respectively.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

See [D3D12_PIPELINE_STATE_STREAM_DESC](#) for a description of the layout and behavior of a streaming pipeline desc.

[ID3D12PipelineLibrary1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineState interface (d3d12.h)

Article 02/22/2024

Represents the state of all currently set shaders as well as certain fixed function state objects.

Inheritance

The **ID3D12PipelineState** interface inherits from [ID3D12Pageable](#). **ID3D12PipelineState** also has these types of members:

Methods

The **ID3D12PipelineState** interface has these methods.

[+] Expand table

[ID3D12PipelineState::GetCachedBlob](#)

Gets the cached blob representing the pipeline state.

Remarks

Use [ID3D12Device::CreateGraphicsPipelineState](#) or [ID3D12Device::CreateComputePipelineState](#) to create a pipeline state object (PSO).

A pipeline state object corresponds to a significant portion of the state of the graphics processing unit (GPU). This state includes all currently set shaders and certain fixed function state objects. The only way to change states contained within the pipeline object is to change the currently bound pipeline object.

Examples

The [D3D12DynamicIndexing](#) sample uses **ID3D12PipelineState** as follows:

Declare the pipeline objects.

C++

```

// Asset objects.
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12PipelineState> m_computeState;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
ComPtr<ID3D12Resource> m_vertexBuffer;
ComPtr<ID3D12Resource> m_vertexBufferUpload;
D3D12_VERTEX_BUFFER_VIEW m_vertexBufferView;
ComPtr<ID3D12Resource> m_particleBuffer0[ThreadCount];
ComPtr<ID3D12Resource> m_particleBuffer1[ThreadCount];
ComPtr<ID3D12Resource> m_particleBuffer0Upload[ThreadCount];
ComPtr<ID3D12Resource> m_particleBuffer1Upload[ThreadCount];
ComPtr<ID3D12Resource> m_constantBufferGS;
UINT8* m_pConstantBufferGSData;
ComPtr<ID3D12Resource> m_constantBufferCS;

```

Initializing a bundle.

C++

```

void FrameResource::InitBundle(ID3D12Device* pDevice, ID3D12PipelineState*
pPso,
    UINT frameResourceIndex, UINT numIndices, D3D12_INDEX_BUFFER_VIEW*
pIndexBufferViewDesc, D3D12_VERTEX_BUFFER_VIEW* pVertexBufferViewDesc,
    ID3D12DescriptorHeap* pCbvSrvDescriptorHeap, UINT cbvSrvDescriptorSize,
ID3D12DescriptorHeap* pSamplerDescriptorHeap, ID3D12RootSignature*
pRootSignature)
{
    ThrowIfFailed(pDevice->CreateCommandList(0,
D3D12_COMMAND_LIST_TYPE_BUNDLE, m_bundleAllocator.Get(), pPso,
IID_PPV_ARGS(&m_bundle)));

    PopulateCommandList(m_bundle.Get(), pPso, frameResourceIndex,
numIndices, pIndexBufferViewDesc,
        pVertexBufferViewDesc, pCbvSrvDescriptorHeap, cbvSrvDescriptorSize,
pSamplerDescriptorHeap, pRootSignature);

    ThrowIfFailed(m_bundle->Close());
}

```

The [D3D12Bundles](#) sample uses **ID3D12PipelineState** as follows:

Populating the command lists, note the alternating PSO.

C++

```

void FrameResource::PopulateCommandList(ID3D12GraphicsCommandList*
pCommandList, ID3D12PipelineState* pPso1, ID3D12PipelineState* pPso2,
    UINT frameResourceIndex, UINT numIndices, D3D12_INDEX_BUFFER_VIEW*

```

```

pIndexBufferViewDesc, D3D12_VERTEX_BUFFER_VIEW* pVertexBufferViewDesc,
    ID3D12DescriptorHeap* pCbvSrvDescriptorHeap, UINT cbvSrvDescriptorSize,
    ID3D12DescriptorHeap* pSamplerDescriptorHeap, ID3D12RootSignature*
pRootSignature)
{
    // If the root signature matches the root signature of the caller, then
    // bindings are inherited, otherwise the bind space is reset.
    pCommandList->SetGraphicsRootSignature(pRootSignature);

    ID3D12DescriptorHeap* ppHeaps[] = { pCbvSrvDescriptorHeap,
    pSamplerDescriptorHeap };
    pCommandList->SetDescriptorHeaps(_countof(ppHeaps), ppHeaps);
    pCommandList-
>IASetPrimitiveTopology(D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST);

    pCommandList->IASetIndexBuffer(pIndexBufferViewDesc);

    pCommandList->IASetVertexBuffers(0, 1, pVertexBufferViewDesc);

    pCommandList->SetGraphicsRootDescriptorTable(0, pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());
    pCommandList->SetGraphicsRootDescriptorTable(1, pSamplerDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart());

    // Calculate the descriptor offset due to multiple frame resources.
    // 1 SRV + how many CBVs we have currently.
    UINT frameResourceDescriptorOffset = 1 + (frameResourceIndex *
m_cityRowCount * m_cityColumnCount);
    CD3DX12_GPU_DESCRIPTOR_HANDLE cbvSrvHandle(pCbvSrvDescriptorHeap-
>GetGPUDescriptorHandleForHeapStart(), frameResourceDescriptorOffset,
cbvSrvDescriptorSize);

    BOOL usePso1 = TRUE;
    for (UINT i = 0; i < m_cityRowCount; i++)
    {
        for (UINT j = 0; j < m_cityColumnCount; j++)
        {
            // Alternate which PSO to use; the pixel shader is different on
            // each just as a PSO setting demonstration.
            pCommandList->SetPipelineState(usePso1 ? pPso1 : pPso2);
            usePso1 = !usePso1;

            // Set this city's CBV table and move to the next descriptor.
            pCommandList->SetGraphicsRootDescriptorTable(2, cbvSrvHandle);
            cbvSrvHandle.Offset(cbvSrvDescriptorSize);

            pCommandList->DrawIndexedInstanced(numIndices, 1, 0, 0, 0);
        }
    }
}

```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12PipelineState::GetCachedBlob method (d3d12.h)

Article 02/22/2024

Gets the cached blob representing the pipeline state.

Syntax

C++

```
HRESULT GetCachedBlob(  
    [out] ID3DBlob **ppBlob  
) ;
```

Parameters

[out] ppBlob

Type: **ID3DBlob****

After this method returns, points to the cached blob representing the pipeline state.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Refer to the remarks for [D3D12_CACHED_PIPELINE_STATE](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineState](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedResourceSession interface (d3d12.h)

Article 02/22/2024

Monitors the validity of a protected resource session. This interface extends [ID3D12ProtectedSession](#).

You can obtain an **ID3D12ProtectedResourceSession** by calling [ID3D12Device4::CreateProtectedResourceSession](#).

Inheritance

The **ID3D12ProtectedResourceSession** interface inherits from the **ID3D12ProtectedSession** interface.

Methods

The **ID3D12ProtectedResourceSession** interface has these methods.

[+] Expand table

[ID3D12ProtectedResourceSession::GetDesc](#)

Retrieves a description of the protected resource session.
([ID3D12ProtectedResourceSession.GetDesc](#))

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

- ID3D12ProtectedSession
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedResourceSession::GetDesc method (d3d12.h)

Article02/22/2024

Retrieves a description of the protected resource session.

Syntax

C++

```
D3D12_PROTECTED_RESOURCE_SESSION_DESC GetDesc();
```

Return value

A [D3D12_PROTECTED_RESOURCE_SESSION_DESC](#) that describes the protected resource session.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12ProtectedResourceSession](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedResourceSession1 interface (d3d12.h)

Article 02/22/2024

Monitors the validity of a protected resource session. This interface extends [ID3D12ProtectedSession](#).

You can obtain an **ID3D12ProtectedResourceSession1** by calling [ID3D12Device7::CreateProtectedResourceSession1](#).

Inheritance

The **ID3D12ProtectedResourceSession1** interface inherits from the **ID3D12ProtectedResourceSession** interface.

Methods

The **ID3D12ProtectedResourceSession1** interface has these methods.

[+] Expand table

[ID3D12ProtectedResourceSession1::GetDesc1](#)

Retrieves a description of the protected resource session.
([ID3D12ProtectedResourceSession1::GetDesc1](#))

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

- [ID3D12ProtectedResourceSession](#)
-

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedResourceSession1::GetDesc1 method (d3d12.h)

Article02/22/2024

Retrieves a description of the protected resource session.

Syntax

C++

```
D3D12_PROTECTED_RESOURCE_SESSION_DESC1 GetDesc1();
```

Return value

A [D3D12_PROTECTED_RESOURCE_SESSION_DESC1](#) that describes the protected resource session.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [ID3D12ProtectedResourceSession1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedSession interface (d3d12.h)

Article 02/22/2024

Offers base functionality that allows for a consistent way to monitor the validity of a session across the different types of sessions. The only type of session currently available is of type [ID3D12ProtectedResourceSession](#).

Inheritance

The **ID3D12ProtectedSession** interface inherits from [ID3D12DeviceChild](#).

ID3D12ProtectedSession also has these types of members:

Methods

The **ID3D12ProtectedSession** interface has these methods.

[+] Expand table

ID3D12ProtectedSession::GetSessionStatus	Gets the status of the protected session.
ID3D12ProtectedSession::GetStatusFence	Retrieves the fence for the protected session. From the fence, you can retrieve the current uniqueness validity value (using <code>ID3D12Fence::GetCompletedValue</code>), and add monitors for changes to its value. This is a read-only fence.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12DeviceChild](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedSession::GetSessionStatus method (d3d12.h)

Article02/22/2024

Gets the status of the protected session.

Syntax

C++

```
D3D12_PROTECTED_SESSION_STATUS GetSessionStatus();
```

Return value

Type: [D3D12_PROTECTED_SESSION_STATUS](#)

The status of the protected session. If the returned value is [D3D12_PROTECTED_SESSION_STATUS_INVALID](#), then you need to wait for a uniqueness value bump to reuse the resource if the session is an [ID3D12ProtectedResourceSession](#).

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ProtectedSession::GetStatusFence method (d3d12.h)

Article02/22/2024

Retrieves the fence for the protected session. From the fence, you can retrieve the current uniqueness validity value (using [ID3D12Fence::GetCompletedValue](#)), and add monitors for changes to its value. This is a read-only fence.

Syntax

C++

```
HRESULT GetStatusFence(
    REFIID riid,
    [optional] void    **ppFence
);
```

Parameters

riid

The GUID of the interface to a fence. Most commonly, [ID3D12Fence](#), although it may be any GUID for any interface. If the protected session object doesn't support the interface for this GUID, the function returns [E_NOINTERFACE](#).

[optional] ppFence

A pointer to a memory block that receives a pointer to the fence for the given protected session.

Return value

If this method succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT](#) error code.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12ProtectedSession](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12QueryHeap interface (d3d12.h)

Article 02/22/2024

Manages a query heap. A query heap holds an array of queries, referenced by indexes.

Inheritance

The **ID3D12QueryHeap** interface inherits from the **ID3D12Pageable** interface.

Remarks

For more information, refer to [Queries](#).

Examples

The [D3D12PredicationQueries](#) sample uses **ID3D12QueryHeap** as follows:

Create a query heap and a query result buffer.

C++

```
// Pipeline objects.
D3D12_VIEWPORT m_viewport;
D3D12_RECT m_scissorRect;
ComPtr<IDXGISwapChain3> m_swapChain;
ComPtr<ID3D12Device> m_device;
ComPtr<ID3D12Resource> m_renderTargets[FrameCount];
ComPtr<ID3D12CommandAllocator> m_commandAllocators[FrameCount];
ComPtr<ID3D12CommandQueue> m_commandQueue;
ComPtr<ID3D12RootSignature> m_rootSignature;
ComPtr<ID3D12DescriptorHeap> m_rtvHeap;
ComPtr<ID3D12DescriptorHeap> m_cbvHeap;
ComPtr<ID3D12DescriptorHeap> m_dsvHeap;
ComPtr<ID3D12QueryHeap> m_queryHeap;
UINT m_rtvDescriptorSize;
UINT m_cbvSrvDescriptorSize;
UINT m_frameIndex;

// Synchronization objects.
ComPtr<ID3D12Fence> m_fence;
UINT64 m_fenceValues[FrameCount];
HANDLE m_fenceEvent;

// Asset objects.
ComPtr<ID3D12PipelineState> m_pipelineState;
ComPtr<ID3D12PipelineState> m_queryState;
```

```
ComPtr<ID3D12GraphicsCommandList> m_commandList;
ComPtr<ID3D12Resource> m_vertexBuffer;
ComPtr<ID3D12Resource> m_constantBuffer;
ComPtr<ID3D12Resource> m_depthStencil;
ComPtr<ID3D12Resource> m_queryResult;
D3D12_VERTEX_BUFFER_VIEW m_vertexBufferView;
```

C++

```
// Describe and create a heap for occlusion queries.
D3D12_QUERY_HEAP_DESC queryHeapDesc = {};
queryHeapDesc.Count = 1;
queryHeapDesc.Type = D3D12_QUERY_HEAP_TYPE_OCCLUSION;
ThrowIfFailed(m_device->CreateQueryHeap(&queryHeapDesc,
IID_PPV_ARGS(&m_queryHeap)));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource interface (d3d12.h)

Article 02/22/2024

Encapsulates a generalized ability of the CPU and GPU to read and write to physical memory, or heaps. It contains abstractions for organizing and manipulating simple arrays of data as well as multidimensional data optimized for shader sampling.

Inheritance

The **ID3D12Resource** interface inherits from [ID3D12Pageable](#). **ID3D12Resource** also has these types of members:

Methods

The **ID3D12Resource** interface has these methods.

[] [Expand table](#)

ID3D12Resource::GetDesc
Gets the resource description.
ID3D12Resource::GetGPUVirtualAddress
This method returns the GPU virtual address of a buffer resource.
ID3D12Resource::GetHeapProperties
Retrieves the properties of the resource heap, for placed and committed resources.
ID3D12Resource::Map
Gets a CPU pointer to the specified subresource in the resource, but may not disclose the pointer value to applications. Map also invalidates the CPU cache, when necessary, so that CPU reads to this address reflect any modifications made by the GPU.
ID3D12Resource::ReadFromSubresource
Uses the CPU to copy data from a subresource, enabling the CPU to read the contents of most textures with undefined layouts.
ID3D12Resource::Unmap

Invalidate the CPU pointer to the specified subresource in the resource.

[ID3D12Resource::WriteToSubresource](#)

Uses the CPU to copy data into a subresource, enabling the CPU to modify the contents of most textures with undefined layouts.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::GetDesc method (d3d12.h)

Article 02/22/2024

Gets the resource description.

Syntax

C++

```
D3D12_RESOURCE_DESC GetDesc();
```

Return value

Type: [D3D12_RESOURCE_DESC](#)

A Direct3D 12 resource description structure.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Resource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::GetGPUVirtualAddress method (d3d12.h)

Article 02/22/2024

This method returns the GPU virtual address of a buffer resource.

Syntax

C++

```
D3D12_GPU_VIRTUAL_ADDRESS GetGPUVirtualAddress();
```

Return value

Type: D3D12_GPU_VIRTUAL_ADDRESS

This method returns the GPU virtual address. D3D12_GPU_VIRTUAL_ADDRESS is a typedef'd synonym of UINT64.

Remarks

This method is only useful for buffer resources, it will return zero for all texture resources.

For more information on the use of GPU virtual addresses, refer to [Indirect Drawing](#).

Examples

The [D3D1211on12](#) sample uses **ID3D12Resource::GetGPUVirtualAddress** as follows:

C++

```
// Initialize the vertex buffer view.  
m_vertexBufferView.BufferLocation = m_vertexBuffer->GetGPUVirtualAddress();  
m_vertexBufferView.StrideInBytes = sizeof(Vertex);  
m_vertexBufferView.SizeInBytes = vertexBufferSize;
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12Resource::GetHeapProperties method (d3d12.h)

Article 02/22/2024

Retrieves the properties of the resource heap, for placed and committed resources.

Syntax

C++

```
HRESULT GetHeapProperties(
    [out, optional] D3D12_HEAP_PROPERTIES *pHeapProperties,
    [out, optional] D3D12_HEAP_FLAGS      *pHeapFlags
);
```

Parameters

[out, optional] pHeapProperties

Type: [D3D12_HEAP_PROPERTIES*](#)

Pointer to a [D3D12_HEAP_PROPERTIES](#) structure, that on successful completion of the method will contain the resource heap properties.

[out, optional] pHeapFlags

Type: [D3D12_HEAP_FLAGS*](#)

Specifies a [D3D12_HEAP_FLAGS](#) variable, that on successful completion of the method will contain any miscellaneous heap flags.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#). If the resource was created as reserved, E_INVALIDARG is returned.

Remarks

This method only works on placed and committed resources, not on reserved resources. If the resource was created as reserved, E_INVALIDARG is returned. The pages could be mapped to none, one, or more heaps.

For more information, refer to [Memory Management in Direct3D 12](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::Map method (d3d12.h)

Article 08/19/2022

Gets a CPU pointer to the specified subresource in the resource, but may not disclose the pointer value to applications. **Map** also invalidates the CPU cache, when necessary, so that CPU reads to this address reflect any modifications made by the GPU.

Syntax

C++

```
HRESULT Map(
    UINT             Subresource,
    [in, optional]  const D3D12_RANGE *pReadRange,
    [out, optional] void        **ppData
);
```

Parameters

Subresource

Type: **UINT**

Specifies the index number of the subresource.

[in, optional] pReadRange

Type: **const D3D12_RANGE***

A pointer to a **D3D12_RANGE** structure that describes the range of memory to access.

This indicates the region the CPU might read, and the coordinates are subresource-relative. A null pointer indicates the entire subresource might be read by the CPU. It is valid to specify the CPU won't read any data by passing a range where **End** is less than or equal to **Begin**.

[out, optional] ppData

Type: **void****

A pointer to a memory block that receives a pointer to the resource data.

A null pointer is valid and is useful to cache a CPU virtual address range for methods like [WriteToSubresource](#). When *ppData* is not NULL, the pointer returned is never offset by any values in *pReadRange*.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Map and [Unmap](#) can be called by multiple threads safely. Nested **Map** calls are supported and are ref-counted. The first call to **Map** allocates a CPU virtual address range for the resource. The last call to [Unmap](#) deallocates the CPU virtual address range. The CPU virtual address is commonly returned to the application; but manipulating the contents of textures with unknown layouts precludes disclosing the CPU virtual address. See [WriteToSubresource](#) for more details. Applications cannot rely on the address being consistent, unless **Map** is persistently nested.

Pointers returned by **Map** are not guaranteed to have all the capabilities of normal pointers, but most applications won't notice a difference in normal usage. For example, pointers with WRITE_COMBINE behavior have weaker CPU memory ordering guarantees than WRITE_BACK behavior. Memory accessible by both CPU and GPU are not guaranteed to share the same atomic memory guarantees that the CPU has, due to PCIe limitations. Use fences for synchronization.

There are two usage model categories for **Map**, simple and advanced. The simple usage models maximize tool performance, so applications are recommended to stick with the simple models until the advanced models are proven to be required by the app.

Simple Usage Models

Applications should stick to the heap type abstractions of UPLOAD, DEFAULT, and READBACK, in order to support all adapter architectures reasonably well.

Applications should avoid CPU reads from pointers to resources on UPLOAD heaps, even accidentally. CPU reads will work, but are prohibitively slow on many common GPU architectures, so consider the following:

- Don't make the CPU read from resources associated with heaps that are D3D12_HEAP_TYPE_UPLOAD or have

D3D12_CPU_PROPERTY_WRITE_COMBINE.

- The memory region to which **pData** points can be allocated with [PAGE_WRITECOMBINE](#), and your app must honor all restrictions that are associated with such memory.
- Even the following C++ code can read from memory and trigger the performance penalty because the code can expand to the following x86 assembly code.

C++ code:

```
*((int*)MappedResource.pData) = 0;
```

x86 assembly code:

```
AND DWORD PTR [EAX],0
```

- Use the appropriate optimization settings and language constructs to help avoid this performance penalty. For example, you can avoid the xor optimization by using a **volatile** pointer or by optimizing for code speed instead of code size.

Applications are encouraged to leave resources unmapped while the CPU will not modify them, and use tight, accurate ranges at all times. This enables the fastest modes for tools, like [Graphics Debugging](#) and the debug layer. Such tools need to track all CPU modifications to memory that the GPU could read.

Advanced Usage Models

Resources on CPU-accessible heaps can be persistently mapped, meaning **Map** can be called once, immediately after resource creation. **Unmap** never needs to be called, but the address returned from **Map** must no longer be used after the last reference to the resource is released. When using persistent map, the application must ensure the CPU finishes writing data into memory before the GPU executes a command list that reads or writes the memory. In common scenarios, the application merely must write to memory before calling [ExecuteCommandLists](#); but using a fence to delay command list execution works as well.

All CPU-accessible memory types support persistent mapping usage, where the resource is mapped but then never unmapped, provided the application does not access the pointer after the resource has been disposed.

Examples

The D3D12Bundles sample uses `ID3D12Resource::Map` as follows:

Copy triangle data to the vertex buffer.

C++

```
// Copy the triangle data to the vertex buffer.  
UINT8* pVertexDataBegin;  
CD3DX12_RANGE readRange(0, 0);           // We do not intend to read from this  
                                         // resource on the CPU.  
ThrowIfFailed(m_vertexBuffer->Map(0, &readRange, reinterpret_cast<void**>  
(&pVertexDataBegin));  
memcpy(pVertexDataBegin, triangleVertices, sizeof(triangleVertices));  
m_vertexBuffer->Unmap(0, nullptr);
```

Create an upload heap for the constant buffers.

C++

```
// Create an upload heap for the constant buffers.  
ThrowIfFailed(pDevice->CreateCommittedResource(  
    &CD3DX12_HEAP_PROPERTIES(D3D12_HEAP_TYPE_UPLOAD),  
    D3D12_HEAP_FLAG_NONE,  
    &CD3DX12_RESOURCE_DESC::Buffer(sizeof(ConstantBuffer) * m_cityRowCount *  
m_cityColumnCount),  
    D3D12_RESOURCE_STATE_GENERIC_READ,  
    nullptr,  
    IID_PPV_ARGS(&m_cbvUploadHeap));  
  
// Map the constant buffers. Note that unlike D3D11, the resource  
// does not need to be unmapped for use by the GPU. In this sample,  
// the resource stays 'permanently' mapped to avoid overhead with  
// mapping/unmapping each frame.  
CD3DX12_RANGE readRange(0, 0);           // We do not intend to read from this  
                                         // resource on the CPU.  
ThrowIfFailed(m_cbvUploadHeap->Map(0, &readRange, reinterpret_cast<void**>  
(&m_pConstantBuffers)));
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Resource](#)

[Subresources](#)

[Unmap](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::ReadFromSubresource method (d3d12.h)

Article 02/22/2024

Uses the CPU to copy data from a subresource, enabling the CPU to read the contents of most textures with undefined layouts.

Syntax

C++

```
HRESULT ReadFromSubresource(  
    [out] void* pDstData,  
    UINT DstRowPitch,  
    UINT DstDepthPitch,  
    UINT SrcSubresource,  
    [in, optional] const D3D12_BOX *pSrcBox  
);
```

Parameters

[out] pDstData

Type: **void***

A pointer to the destination data in memory.

DstRowPitch

Type: **UINT**

The distance from one row of destination data to the next row.

DstDepthPitch

Type: **UINT**

The distance from one depth slice of destination data to the next.

SrcSubresource

Type: **UINT**

Specifies the index of the subresource to read from.

[in, optional] pSrcBox

Type: [const D3D12_BOX*](#)

A pointer to a box that defines the portion of the destination subresource to copy the resource data from. If NULL, the data is read from the destination subresource with no offset. The dimensions of the destination must fit the destination (see [D3D12_BOX](#)).

An empty box results in a no-op. A box is empty if the top value is greater than or equal to the bottom value, or the left value is greater than or equal to the right value, or the front value is greater than or equal to the back value. When the box is empty, this method doesn't perform any operation.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

See the Remarks section for [WriteToSubresource](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::Unmap method (d3d12.h)

Article 02/22/2024

Invalidate the CPU pointer to the specified subresource in the resource.

Syntax

C++

```
void Unmap(
    UINT Subresource,
    [in, optional] const D3D12_RANGE *pWrittenRange
);
```

Parameters

Subresource

Type: **UINT**

Specifies the index of the subresource.

[in, optional] pWrittenRange

Type: **const D3D12_RANGE***

A pointer to a **D3D12_RANGE** structure that describes the range of memory to unmap.

This indicates the region the CPU might have modified, and the coordinates are subresource-relative. A null pointer indicates the entire subresource might have been modified by the CPU. It is valid to specify the CPU didn't write any data by passing a range where **End** is less than or equal to **Begin**.

This parameter is only used by tooling, and not for correctness of the actual unmap operation.

Return value

None

Remarks

Refer to the extensive Remarks and Examples for the [Map](#) method.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Resource](#)

[Map](#)

[Subresources](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource::WriteToSubresource method (d3d12.h)

Article 10/13/2021

Uses the CPU to copy data into a subresource, enabling the CPU to modify the contents of most textures with undefined layouts.

Syntax

C++

```
HRESULT WriteToSubresource(
    UINT             DstSubresource,
    [in, optional] const D3D12_BOX *pDstBox,
    [in]           const void     *pSrcData,
    UINT             SrcRowPitch,
    UINT             SrcDepthPitch
);
```

Parameters

DstSubresource

Type: **UINT**

Specifies the index of the subresource.

[in, optional] pDstBox

Type: **const D3D12_BOX***

A pointer to a box that defines the portion of the destination subresource to copy the resource data into. If NULL, the data is written to the destination subresource with no offset. The dimensions of the source must fit the destination (see [D3D12_BOX](#)).

An empty box results in a no-op. A box is empty if the top value is greater than or equal to the bottom value, or the left value is greater than or equal to the right value, or the front value is greater than or equal to the back value. When the box is empty, this method doesn't perform any operation.

[in] pSrcData

Type: **const void***

A pointer to the source data in memory.

SrcRowPitch

Type: **UINT**

The distance from one row of source data to the next row.

SrcDepthPitch

Type: **UINT**

The distance from one depth slice of source data to the next.

Return value

Type: **HRESULT**

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

The resource should first be mapped using [Map](#). Textures must be in the [D3D12_RESOURCE_STATE_COMMON](#) state for CPU access through [WriteToSubresource](#) and [ReadFromSubresource](#) to be legal; but buffers do not.

For efficiency, ensure the bounds and alignment of the extents within the box are (64 / [bytes per pixel]) pixels horizontally. Vertical bounds and alignment should be 2 rows, except when 1-byte-per-pixel formats are used, in which case 4 rows are recommended. Single depth slices per call are handled efficiently. It is recommended but not necessary to provide pointers and strides which are 128-byte aligned.

When writing to sub mipmap levels, it is recommended to use larger width and heights than described above. This is because small mipmap levels may actually be stored within a larger block of memory, with an opaque amount of offsetting which can interfere with alignment to cache lines.

[WriteToSubresource](#) and [ReadFromSubresource](#) enable near zero-copy optimizations for UMA adapters, but can prohibitively impair the efficiency of discrete/ NUMA adapters as the texture data cannot reside in local video memory. Typical applications should stick to discrete-friendly upload techniques, unless they recognize the adapter

architecture is UMA. For more details on uploading, refer to [CopyTextureRegion](#), and for more details on UMA, refer to [D3D12_FEATURE_DATA_ARCHITECTURE](#).

On UMA systems, this routine can be used to minimize the cost of memory copying through the loop optimization known as [loop tiling](#). By breaking up the upload into chunks that comfortably fit in the CPU cache, the effective bandwidth between the CPU and main memory more closely achieves theoretical maximums.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

[Subresources](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource1 interface (d3d12.h)

Article02/22/2024

Inheritance

The ID3D12Resource1 interface inherits from the ID3D12Resource interface.

Methods

The ID3D12Resource1 interface has these methods.

 Expand table

ID3D12Resource1::GetProtectedResourceSession

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource1::GetProtectedResourceSession method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetProtectedResourceSession(  
    REFIID riid,  
    void    **ppProtectedSession  
>;
```

Parameters

riid

ppProtectedSession

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource2 interface (d3d12.h)

Article02/22/2024

Inheritance

The ID3D12Resource2 interface inherits from the ID3D12Resource1 interface.

Methods

The ID3D12Resource2 interface has these methods.

 Expand table

ID3D12Resource2::GetDesc1

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Resource2::GetDesc1 method (d3d12.h)

Article02/22/2024

Syntax

C++

```
D3D12_RESOURCE_DESC1 GetDesc1();
```

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12RootSignature interface (d3d12.h)

Article 02/22/2024

The root signature defines what resources are bound to the graphics pipeline. A root signature is configured by the app and links command lists to the resources the shaders require. Currently, there is one graphics and one compute root signature per app.

Inheritance

The **ID3D12RootSignature** interface inherits from the **ID3D12DeviceChild** interface.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12DeviceChild](#)

[Root Signatures](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12RootSignatureDeserializer interface (d3d12.h)

Article 02/22/2024

Contains a method to return the deserialized [D3D12_ROOT_SIGNATURE_DESC](#) data structure, of a serialized root signature version 1.0.

Inheritance

The **ID3D12RootSignatureDeserializer** interface inherits from the [IUnknown](#) interface. **ID3D12RootSignatureDeserializer** also has these types of members:

Methods

The **ID3D12RootSignatureDeserializer** interface has these methods.

[+] Expand table

<p>ID3D12RootSignatureDeserializer::GetRootSignatureDesc</p> <p>Gets the layout of the root signature.</p>

Remarks

This interface has been superceded by [ID3D12VersionedRootSignatureDeserializer](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

[Root Signatures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12RootSignatureDeserializer::GetRootSignatureDesc method (d3d12.h)

Article 02/22/2024

Gets the layout of the root signature.

Syntax

C++

```
const D3D12_ROOT_SIGNATURE_DESC * GetRootSignatureDesc();
```

Return value

Type: [D3D12_ROOT_SIGNATURE_DESC](#)

This method returns a deserialized root signature in a [D3D12_ROOT_SIGNATURE_DESC](#) structure that describes the layout of the root signature.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12RootSignatureDeserializer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12SDKConfiguration interface (d3d12.h)

Article 02/16/2023

Provides SDK configuration methods. A pointer to this interface can be retrieved by calling the [D3D12GetInterface](#) free function with the **CLSID_D3D12SDKConfiguration** CLSID.

Inheritance

The **ID3D12SDKConfiguration** interface inherits from the **IUnknown** interface.

Methods

The **ID3D12SDKConfiguration** interface has these methods.

[+] [Expand table](#)

<p>ID3D12SDKConfiguration::SetSDKVersion</p> <p>Configures the SDK version to use.</p>

Remarks

Tools that play back API capture such as PIX, and test harnesses such as the HLK, require modification to support the redist. Such tools can choose to ship with the latest redist. Direct3D's API compatibility through updates should mean that an API capture tool can capture on an older version of the Direct3D 12 SDK, and play it back on the newer version. However, some scenarios require more flexibility in selecting the SDK version.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348

Requirement	Supported server
Target Platform	Windows 10 Build 20348
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12SDKConfiguration::SetSDKVersion method (d3d12.h)

Article 02/22/2024

Configures the SDK version to use.

Syntax

C++

```
HRESULT SetSDKVersion(  
    UINT    SDKVersion,  
    LPCSTR  SDKPath  
) ;
```

Parameters

SDKVersion

Type: [UINT](#)

The SDK version to set.

SDKPath

Type: [_In_z_ LPCSTR](#)

A NULL-terminated string that provides the relative path to [d3d12core.dll](#) at the specified *SDKVersion*. The path is relative to the process exe of the caller. If [d3d12core.dll](#) isn't found, or isn't of the specified *SDKVersion*, then Direct3D 12 device creation fails.

Return value

Type: [HRESULT](#)

If the function succeeds, then it returns [S_OK](#). Otherwise, it returns one of the [Direct3D 12 return codes](#).

Remarks

This method can be used only in Windows Developer Mode.

To set the SDK version using this API, you must call it before you create the Direct3D 12 device. Calling this API *after* creating the Direct3D 12 device will cause the Direct3D 12 runtime to remove the device.

If the `d3d12core.dll` installed with the OS is newer than the SDK version specified, then the OS version is used instead.

You can retrieve the version of a particular `D3D12Core.dll` from the exported symbol [D3D12SDKVersion](#), which is a variable of type `UINT`, just like the variables exported from applications to enable use of the Agility SDK.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	<code>d3d12.h</code>
Library	<code>D3D12.lib</code>
DLL	<code>D3D12.dll</code>

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderCacheSession interface (d3d12.h)

Article 02/22/2024

Represents a shader cache session.

Inheritance

The **ID3D12ShaderCacheSession** interface inherits from the [ID3D12DeviceChild](#) interface.

Methods

The **ID3D12ShaderCacheSession** interface has these methods.

[+] Expand table

ID3D12ShaderCacheSession::FindValue	Looks up an entry in the cache whose key exactly matches the provided key.
ID3D12ShaderCacheSession::GetDesc	Retrieves the description used to create the cache session.
ID3D12ShaderCacheSession::SetDeleteOnDestroy	When all cache session objects corresponding to a given cache are destroyed, the cache is cleared.
ID3D12ShaderCacheSession::StoreValue	Adds an entry to the cache.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348

Requirement	Value
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12.h

See also

- [ID3D12DeviceChild](#)
- [D3D12 shader cache APIs ↗](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12ShaderCacheSession::FindValue method (d3d12.h)

Article 02/22/2024

Looks up an entry in the cache whose key exactly matches the provided key.

Call the function twice. The first time to retrieve the value's size, and the second time to retrieve the data. In-memory temporary storage makes this calling pattern performant.

Syntax

C++

```
HRESULT FindValue(
    const void *pKey,
    UINT          KeySize,
    void          *pValue,
    UINT          *pValueSize
);
```

Parameters

pKey

Type: `_In_reads_bytes_(KeySize) const void *`

The key of the entry to look up.

KeySize

Type: `UINT`

The size of the key, in bytes.

pValue

Type: `_Out_writes_bytes_(*pValueSize) void *`

A pointer to a memory block that receives the cached entry.

pValueSize

Type: `_Inout_ UINT*`

A pointer to a **UINT** that receives the size of the cached entry, in bytes.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns **S_OK**. Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
<code>DXGI_ERROR_CACHE_HASH_COLLISION</code>	There's an entry with the same hash as the provided key, but the key doesn't exactly match.
<code>DXGI_ERROR_NOT_FOUND</code>	The entry isn't present.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderCacheSession::GetDesc method (d3d12.h)

Article02/22/2024

Retrieves the description used to create the cache session.

Syntax

C++

```
D3D12_SHADER_CACHE_SESSION_DESC GetDesc();
```

Return value

A [D3D12_SHADER_CACHE_SESSION_DESC](#) structure representing the description used to create the cache session.

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs ↗](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderCacheSession::SetDeleteOnDestroy method (d3d12.h)

Article02/22/2024

When all cache session objects corresponding to a given cache are destroyed, the cache is cleared.

See Remarks for the ways in which a disk cache can be cleared.

Syntax

C++

```
void SetDeleteOnDestroy();
```

Return value

None

Remarks

A disk cache can be cleared in one of the following ways.

- Explicitly, by calling [SetDeleteOnDestroy](#) on the session object, and then releasing the session.
- Explicitly, in developer mode, by calling [ID3D12Device9::ShaderCacheControl](#) with [D3D12_SHADER_CACHE_KIND_FLAG_APPLICATION_MANAGED](#).
- Implicitly, by creating a session object with a version that doesn't match the version used to create it.
- Externally, by the disk cleanup utility enumerating it and clearing it. This won't happen for caches created with the [D3D12_SHADER_CACHE_FLAG_USE_WORKING_DIR](#) flag.
- Manually, by deleting the files (*.idx, *.val, and *.lock) stored on disk for [D3D12_SHADER_CACHE_FLAG_USE_WORKING_DIR](#) caches. Your application shouldn't attempt to do this for caches stored outside of the working directory.

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs](#)

Feedback

Was this page helpful?

[!\[\]\(1eb5ad4c78603ab4e11bc056d83bbaed_img.jpg\) Yes](#)[!\[\]\(a039ec0553f857e124068e48ff0bebdb_img.jpg\) No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderCacheSession::StoreValue method (d3d12.h)

Article 02/22/2024

Adds an entry to the cache.

Syntax

C++

```
HRESULT StoreValue(
    const void *pKey,
    UINT         KeySize,
    const void *pValue,
    UINT         ValueSize
);
```

Parameters

pKey

Type: `_In_reads_bytes_(KeySize) const void *`

The key of the entry to add.

KeySize

Type: `UINT`

The size of the key, in bytes.

pValue

Type: `_In_reads_bytes_(ValueSize) void *`

A pointer to a memory block containing the entry to add.

ValueSize

Type: `UINT`

The size of the entry to add, in bytes.

Return value

Type: [HRESULT](#)

If the function succeeds, it returns [S_OK](#). Otherwise, it returns an [HRESULT error code](#).

[+] Expand table

Return value	Description
DXGI_ERROR_ALREADY_EXISTS	There's an entry with the same key.
DXGI_ERROR_CACHE_HASH_COLLISION	There's an entry with the same hash as the provided key, but the key doesn't match.
DXGI_ERROR_CACHE_FULL	Adding this entry would cause the cache to become larger than its maximum size.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

- [D3D12 shader cache APIs](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12StateObject interface (d3d12.h)

Article02/22/2024

Represents a variable amount of configuration state, including shaders, that an application manages as a single unit and which is given to a driver atomically to process, such as compile or optimize. Create a state object by calling [ID3D12Device5::CreateStateObject](#).

Inheritance

The **ID3D12StateObject** interface inherits from the **ID3D12Pageable** interface.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Pageable](#)

Feedback

Was this page helpful?

 Yes No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12StateObjectProperties interface (d3d12.h)

Article 02/22/2024

Provides methods for getting and setting the properties of an [ID3D12StateObject](#). To retrieve an instance of this type, call [ID3D12StateObject::QueryInterface](#) with the IID of [ID3D12StateObjectProperties](#).

Inheritance

The [ID3D12StateObjectProperties](#) interface inherits from the [IUnknown](#) interface.

[ID3D12StateObjectProperties](#) also has these types of members:

Methods

The [ID3D12StateObjectProperties](#) interface has these methods.

[+] Expand table

ID3D12StateObjectProperties::GetPipelineStackSize
Gets the current pipeline stack size.
ID3D12StateObjectProperties::GetShaderIdentifier
Retrieves the unique identifier for a shader that can be used in a shader record.
ID3D12StateObjectProperties::GetShaderStackSize
Gets the amount of stack memory required to invoke a raytracing shader in HLSL.
ID3D12StateObjectProperties::SetPipelineStackSize
Set the current pipeline stack size.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12StateObjectProperties::GetPipelineStackSize method (d3d12.h)

Article 02/22/2024

Gets the current pipeline stack size.

Syntax

C++

```
UINT64 GetPipelineStackSize();
```

Return value

The current pipeline stack size in bytes. When called on non-executable state objects, such as collections, the return value is 0.

Remarks

This method and [SetPipelineStackSize](#) are not re-entrant. This means if calling either or both from separate threads, the app must synchronize on its own.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12StateObjectProperties::GetShaderIdentifier method (d3d12.h)

Article02/22/2024

Retrieves the unique identifier for a shader that can be used in a shader record.

Syntax

C++

```
void * GetShaderIdentifier(  
    LPCWSTR pExportName  
);
```

Parameters

pExportName

Entry point in the state object for which to retrieve an identifier.

Return value

A pointer to the shader identifier.

The data referenced by this pointer is valid as long as the state object it came from is valid. The size of the data returned is [D3D12_SHADER_IDENTIFIER_SIZE_IN_BYTES](#).

Applications should copy and cache this data to avoid the cost of searching for it in the state object if it will need to be retrieved many times. The identifier is used in shader records within shader tables in GPU memory, which the app must populate.

The data itself globally identifies the shader, so even if the shader appears in a different state object with same associations, like any root signatures, it will have the same identifier.

If the shader isn't fully resolved in the state object, the return value is `nullptr`.

Requirements

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12StateObjectProperties::GetShaderStackSize method (d3d12.h)

Article 02/22/2024

Gets the amount of stack memory required to invoke a raytracing shader in HLSL.

Syntax

C++

```
UINT64 GetShaderStackSize(  
    LPCWSTR pExportName  
) ;
```

Parameters

pExportName

The shader entrypoint in the state object for which to retrieve stack size. For hit groups, an individual shader within the hit group must be specified using the syntax:

hitGroupName::shaderType

Where *hitGroupName* is the entrypoint name for the hit group and *shaderType* is one of:

- intersection
- anyhit
- closesthit

These values are all case-sensitive.

An example value is: "myTreeLeafHitGroup::anyhit".

Return value

Amount of stack memory, in bytes, required to invoke the shader. If the shader isn't fully resolved in the state object, or the shader is unknown or of a type for which a stack size isn't relevant, such as a hit group, the return value is 0xffffffff. The 32-bit 0xffffffff value is used for the `UINT64` return value to ensure that bad return values don't get lost when summed up with other values as part of calculating an overall pipeline stack size.

Remarks

This method only needs to be called if the app wants to configure the stack size by calling [SetPipelineStackSize](#), rather than relying on the conservative default stack size.

This method is only valid for ray generation shaders, hit groups, miss shaders, and callable shaders. Even ray generation shaders may return a non-zero value despite being at the bottom of the stack.

For hit groups, stack size must be queried for the individual shaders comprising it (intersection shaders, any hit shaders, closest hit shaders), as each likely has a different stack size requirement. The stack size can't be queried on these individual shaders directly, as the way they are compiled can be influenced by the overall hit group that contains them. The *pExportName* parameter includes syntax for identifying individual shaders within a hit group.

This API can be called on either collection state objects or raytracing pipeline state objects.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12StateObjectProperties::SetPipelineStackSize method (d3d12.h)

Article 02/22/2024

Set the current pipeline stack size.

Syntax

C++

```
void SetPipelineStackSize(  
    UINT64 PipelineStackSizeInBytes  
);
```

Parameters

PipelineStackSizeInBytes

Stack size in bytes to use during pipeline execution for each shader thread. There can be many thousands of threads in flight at once on the GPU.

If the value is greater than 0xffffffff (the maximum value of a 32-bit UINT) the runtime will drop the call, and the debug layer will print an error, as this is likely the result of summing up invalid stack sizes returned from [GetShaderStackSize](#) called with invalid parameters, which return 0xffffffff. In this case, the previously set stack size, or the default, remains.

Return value

None

Remarks

This method and [GetPipelineStackSize](#) are not re-entrant. This means if calling either or both from separate threads, the app must synchronize on its own.

The runtime drops calls to state objects other than raytracing pipelines, such as collections.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12SwapChainAssistant interface (d3d12.h)

Article02/22/2024

Inheritance

The **ID3D12SwapChainAssistant** interface inherits from the **IUnknown** interface.

Methods

The **ID3D12SwapChainAssistant** interface has these methods.

[+] Expand table

ID3D12SwapChainAssistant::GetCurrentResourceAndCommandQueue
ID3D12SwapChainAssistant::GetLUID
ID3D12SwapChainAssistant::GetSwapChainObject
ID3D12SwapChainAssistant::InsertImplicitSync

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12SwapChainAssistant::GetCurrentResourceAndCommandQueue method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetCurrentResourceAndCommandQueue(
    REFIID riidResource,
    void    **ppvResource,
    REFIID riidQueue,
    void    **ppvQueue
);
```

Parameters

riidResource

ppvResource

riidQueue

ppvQueue

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12SwapChainAssistant::GetLUID method (d3d12.h)

Article 02/22/2024

Syntax

C++

```
LUID GetLUID();
```

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12SwapChainAssistant::GetSwapChainObject method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT GetSwapChainObject(  
    REFIID riid,  
    void    **ppv  
>;
```

Parameters

riid

ppv

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12SwapChainAssistant::InsertImplicitSync method (d3d12.h)

Article 02/22/2024

Syntax

C++

```
HRESULT InsertImplicitSync();
```

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Tools interface (d3d12.h)

Article 02/22/2024

This interface is used to configure the runtime for tools such as PIX. It's not intended or supported for any other scenario.

Inheritance

The **ID3D12Tools** interface inherits from the [IUnknown](#) interface. **ID3D12Tools** also has these types of members:

Methods

The **ID3D12Tools** interface has these methods.

[] [Expand table](#)

ID3D12Tools::EnableShaderInstrumentation
This method enables tools such as PIX to instrument shaders.
ID3D12Tools::ShaderInstrumentationEnabled
Determines whether shader instrumentation is enabled.

Remarks

Do not use this interface in your application, it's not intended or supported for any scenario other than to enable tooling such as PIX.

Developer Mode must be enabled for this interface to respond.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows

See also

[Core Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Tools::EnableShaderInstrumentation method (d3d12.h)

Article 02/22/2024

This method enables tools such as PIX to instrument shaders.

Syntax

C++

```
void EnableShaderInstrumentation(  
    BOOL bEnable  
)
```

Parameters

bEnable

Type: **BOOL**

TRUE to enable shader instrumentation; otherwise, FALSE.

Return value

None

Remarks

Do not use this interface in your application, it's not intended or supported for any scenario other than to enable tooling such as PIX.

Developer Mode must be enabled for this interface to respond.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Tools](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Tools::ShaderInstrumentationEnabled method (d3d12.h)

Article 02/22/2024

Determines whether shader instrumentation is enabled.

Syntax

C++

```
BOOL ShaderInstrumentationEnabled();
```

Return value

Type: **BOOL**

Returns TRUE if shader instrumentation is enabled; otherwise FALSE.

Remarks

Do not use this interface in your application, it's not intended or supported for any scenario other than to enable tooling such as PIX.

Developer Mode must be enabled for this interface to respond.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Tools](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12VersionedRootSignatureDeserializer interface (d3d12.h)

Article 07/22/2021

Contains methods to return the deserialized [D3D12_ROOT_SIGNATURE_DESC1](#) data structure, of any version of a serialized root signature.

Inheritance

The **ID3D12VersionedRootSignatureDeserializer** interface inherits from the [IUnknown](#) interface. **ID3D12VersionedRootSignatureDeserializer** also has these types of members:

Methods

The **ID3D12VersionedRootSignatureDeserializer** interface has these methods.

[+] Expand table

ID3D12VersionedRootSignatureDeserializer::GetRootSignatureDescAtVersion
Converts root signature description structures to a requested version.
ID3D12VersionedRootSignatureDeserializer::GetUnconvertedRootSignatureDesc
Gets the layout of the root signature, without converting between root signature versions.

Remarks

This interface supercedes [ID3D12RootSignatureDeserializer](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

[Root Signature Version 1.1](#)

[Root Signatures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12VersionedRootSignatureDeserializer::GetRootSignatureDescAtVersion method (d3d12.h)

Article 10/13/2021

Converts root signature description structures to a requested version.

Syntax

C++

```
HRESULT GetRootSignatureDescAtVersion(
    D3D_ROOT_SIGNATURE_VERSION           convertToVersion,
    [out] const D3D12_VERSIONED_ROOT_SIGNATURE_DESC **ppDesc
);
```

Parameters

convertToVersion

Type: [D3D_ROOT_SIGNATURE_VERSION](#)

Specifies the required [D3D_ROOT_SIGNATURE_VERSION](#).

[out] ppDesc

Type: [const D3D12_VERSIONED_ROOT_SIGNATURE_DESC**](#)

Contains the deserialized root signature in a [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#) structure.

Return value

Type: [HRESULT](#)

This method returns an HRESULT success or error code. The method can fail with E_OUTOFMEMORY.

Remarks

This method allocates additional storage if needed for the converted root signature (memory owned by the deserializer interface). If conversion is done, the deserializer interface doesn't free the original serialized root signature memory – all versions the interface has been asked to convert to are available until the deserializer is destroyed.

Converting a root signature from 1.1 to 1.0 will drop all [D3D12_DESCRIPTOR_RANGE_FLAGS](#) and [D3D12_ROOT_DESCRIPTOR_FLAGS](#) can be useful for generating compatible root signatures that need to run on old operating systems, though does lose optimization opportunities. For instance, multiple root signature versions can be serialized and stored with application assets, with the appropriate version used at runtime based on the operating system capabilities.

Converting a root signature from 1.0 to 1.1 just adds the appropriate flags to match 1.0 semantics.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12VersionedRootSignatureDeserializer](#)

[Root Signature Version 1.1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12VersionedRootSignatureDeserializer::GetUnconvertedRootSignatureDesc method (d3d12.h)

Article 02/22/2024

Gets the layout of the root signature, without converting between root signature versions.

Syntax

C++

```
const D3D12_VERSIONED_ROOT_SIGNATURE_DESC *
GetUnconvertedRootSignatureDesc();
```

Return value

Type: [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#)

This method returns a deserialized root signature in a [D3D12_VERSIONED_ROOT_SIGNATURE_DESC](#) structure that describes the layout of the root signature.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12VirtualizationGuestDevice interface (d3d12.h)

Article02/22/2024

TBD

Inheritance

The **ID3D12VirtualizationGuestDevice** interface derives from the [IUnknown](#) interface.

Methods

The **ID3D12VirtualizationGuestDevice** interface has these methods.

[+] Expand table

ID3D12VirtualizationGuestDevice::CreateFenceFd
TBD

ID3D12VirtualizationGuestDevice::ShareWithHost
TBD

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 11, version 22H2
Target Platform	Windows
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12VirtualizationGuestDevice::CreateFenceFd method (d3d12.h)

Article 02/22/2024

Syntax

C++

```
HRESULT CreateFenceFd(
    ID3D12Fence *pFence,
    UINT64      FenceValue,
    int         *pFenceFd
);
```

Parameters

pFence

FenceValue

pFenceFd

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 11, version 22H2
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12VirtualizationGuestDevice::ShareWithHost method (d3d12.h)

Article02/22/2024

Syntax

C++

```
HRESULT ShareWithHost(  
    ID3D12DeviceChild *pObject,  
    HANDLE             *pHandle  
) ;
```

Parameters

pObject

pHandle

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 11, version 22H2
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

Feedback

Was this page helpful?

Yes

No

PFN_D3D12_CREATE_DEVICE callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_CREATE_DEVICE PfnD3d12CreateDevice;

HRESULT PfnD3d12CreateDevice(
    IUnknown *unnamedParam1,
    D3D_FEATURE_LEVEL unnamedParam2,
    REFIID unnamedParam3,
    void **unnamedParam4
)
{...}
```

Parameters

unnamedParam1

unnamedParam2

unnamedParam3

unnamedParam4

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_CREATE_ROOT_SIGNATURE_DESERIALIZER
PfnD3d12CreateRootSignatureDeserializer;

HRESULT PfnD3d12CreateRootSignatureDeserializer(
    LPCVOID pSrcData,
    SIZE_T SrcDataSizeInBytes,
    REFIID pRootSignatureDeserializerInterface,
    void **ppRootSignatureDeserializer
)
{...}
```

Parameters

pSrcData

SrcDataSizeInBytes

pRootSignatureDeserializerInterface

ppRootSignatureDeserializer

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PFN_D3D12_CREATE_VERSIONED_ROOT_SIGNATURE_DESERIALIZER callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_CREATE_VERSIONED_ROOT_SIGNATURE_DESERIALIZER  
PfnD3d12CreateVersionedRootSignatureDeserializer;  
  
HRESULT PfnD3d12CreateVersionedRootSignatureDeserializer(  
    LPCVOID pSrcData,  
    SIZE_T SrcDataSizeInBytes,  
    REFIID pRootSignatureDeserializerInterface,  
    void **ppRootSignatureDeserializer  
)  
{...}
```

Parameters

pSrcData

SrcDataSizeInBytes

pRootSignatureDeserializerInterface

ppRootSignatureDeserializer

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PFN_D3D12_GET_DEBUG_INTERFACE callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_GET_DEBUG_INTERFACE PfnD3d12GetDebugInterface;

HRESULT PfnD3d12GetDebugInterface(
    REFIID unnamedParam1,
    void **unnamedParam2
)
{...}
```

Parameters

unnamedParam1

unnamedParam2

Requirements

[] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PFN_D3D12_GET_INTERFACE callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_GET_INTERFACE PfnD3d12GetInterface;

HRESULT PfnD3d12GetInterface(
    REFCLSID unnamedParam1,
    REFIID unnamedParam2,
    void **unnamedParam3
)
{...}
```

Parameters

unnamedParam1

unnamedParam2

unnamedParam3

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

PFN_D3D12_SERIALIZE_ROOT_SIGNATURE callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_SERIALIZE_ROOT_SIGNATURE PfnD3d12SerializeRootSignature;  
  
HRESULT PfnD3d12SerializeRootSignature(  
    const D3D12_ROOT_SIGNATURE_DESC *pRootSignature,  
    D3D_ROOT_SIGNATURE_VERSION Version,  
    ID3DBlob **ppBlob,  
    ID3DBlob **ppErrorBlob  
)  
{...}
```

Parameters

pRootSignature

Version

ppBlob

ppErrorBlob

Requirements

 Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

PFN_D3D12_SERIALIZE_VERSIONED_ROOT_SIGNATURE callback function (d3d12.h)

Article02/22/2024

Syntax

C++

```
PFN_D3D12_SERIALIZE_VERSIONED_ROOT_SIGNATURE
PfnD3d12SerializeVersionedRootSignature;

HRESULT PfnD3d12SerializeVersionedRootSignature(
    const D3D12_VERSIONED_ROOT_SIGNATURE_DESC *pRootSignature,
    ID3DBlob **ppBlob,
    ID3DBlob **ppErrorBlob
)
{...}
```

Parameters

pRootSignature

ppBlob

ppErrorBlob

Requirements

[+] Expand table

Requirement	Value
Header	d3d12.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

d3d12sdklayers.h header

Article 01/24/2023

This header is used by Direct3D 12 Graphics. For more information, see:

- [Direct3D 12 Graphics](#)

d3d12sdklayers.h contains the following programming interfaces:

Interfaces

ID3D12Debug	An interface used to turn on the debug layer.
ID3D12Debug1	Adds GPU-Based Validation and Dependent Command Queue Synchronization to the debug layer.
ID3D12Debug2	Adds configurable levels of GPU-based validation to the debug layer. (ID3D12Debug2)
ID3D12Debug3	Adds configurable levels of GPU-based validation to the debug layer. (ID3D12Debug3)
ID3D12Debug4	Adds the ability to disable the debug layer.
ID3D12Debug5	Adds to the debug layer the ability to configure the auto-naming of objects.
ID3D12Debug6	A debug interface controls debug settings.
ID3D12DebugCommandList	Provides methods to monitor and debug a command list.

[ID3D12DebugCommandList1](#)

This interface enables modification of additional command list debug layer settings.

[ID3D12DebugCommandQueue](#)

Provides methods to monitor and debug a command queue.

[ID3D12DebugDevice](#)

This interface represents a graphics device for debugging.

[ID3D12DebugDevice1](#)

Specifies device-wide debug layer settings.

[ID3D12InfoQueue](#)

An information-queue interface stores, retrieves, and filters debug messages. The queue consists of a message queue, an optional storage filter stack, and a optional retrieval filter stack.
([ID3D12InfoQueue](#))

[ID3D12SharingContract](#)

Part of a contract between D3D11On12 diagnostic layers and graphics diagnostics tools.

Structures

[D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS](#)

Describes per-command-list settings used by GPU-Based Validation.

[D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#)

Describes settings used by GPU-Based Validation.

[D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR](#)

Describes the amount of artificial slowdown inserted by the debug device to simulate lower-performance graphics adapters.

[D3D12_INFO_QUEUE_FILTER](#)

Debug message filter; contains a lists of message types to allow or deny.
([D3D12_INFO_QUEUE_FILTER](#))

[D3D12_INFO_QUEUE_FILTER_DESC](#)

Allow or deny certain types of messages to pass through a filter.
(D3D12_INFO_QUEUE_FILTER_DESC)

[D3D12_MESSAGE](#)

A debug message in the Information Queue. (D3D12_MESSAGE)

Enumerations

[D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#)

Indicates the debug parameter type used by ID3D12DebugCommandList1::SetDebugParameter and ID3D12DebugCommandList1::GetDebugParameter.

[D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#)

Specifies the data type of the memory pointed to by the pData parameter of ID3D12DebugDevice1::SetDebugParameter and ID3D12DebugDevice1::GetDebugParameter.

[D3D12_DEBUG_FEATURE](#)

Flags for optional D3D12 Debug Layer features.

[D3D12_GPU_BASED_VALIDATION_FLAGS](#)

Describes the level of GPU-based validation to perform at runtime.

[D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS](#)

Specifies how GPU-Based Validation handles patched pipeline states during ID3D12Device::CreateGraphicsPipelineState and ID3D12Device::CreateComputePipelineState.

[D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE](#)

Specifies the type of shader patching used by GPU-Based Validation at either the device or command list level.

[D3D12_MESSAGE_CATEGORY](#)

Specifies categories of debug messages.

D3D12_MESSAGE_ID

Specifies debug message IDs for setting up an info-queue filter (see D3D12_INFO_QUEUE_FILTER); use these messages to allow or deny message categories to pass through the storage and retrieval filters.

D3D12_MESSAGE_SEVERITY

Debug message severity levels for an information queue. (D3D12_MESSAGE_SEVERITY)

D3D12_RLDO_FLAGS

Specifies options for the amount of information to report about a live device object's lifetime.

Feedback

Was this page helpful?



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

D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS structure (d3d12sdklayers.h)

Article 02/22/2024

Describes per-command-list settings used by GPU-Based Validation.

Syntax

C++

```
typedef struct D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS {
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE ShaderPatchMode;
} D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS;
```

Members

ShaderPatchMode

Specifies a [D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE](#) that overrides the default device-level shader patch mode (see [ID3D12DebugDevice1::SetDebugParameter](#)). By default this value is initialized to the *DefaultShaderPatchMode* assigned to the device (see [D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#)).

Remarks

Point to an object using this structure with the *pData* member of [ID3D12DebugCommandList1::SetDebugParameter](#) to configure per-command-list GPU-Based Validation settings.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12sdklayers.h (include D3d12sdklayers_RS1.h)

See also

[Debug Layer Structures](#)

[SetEnableGPUBasedValidation](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEBUG_COMMAND_LIST_PARA METER_TYPE enumeration (d3d12sdklayers.h)

Article02/22/2024

Indicates the debug parameter type used by [ID3D12DebugCommandList1::SetDebugParameter](#) and [ID3D12DebugCommandList1::GetDebugParameter](#).

Syntax

C++

```
typedef enum D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE {
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_GPU_BASED_VALIDATION_SETTINGS = 0
};
```

Constants

[] [Expand table](#)

<code>D3D12_DEBUG_COMMAND_LIST_PARAMETER_GPU_BASED_VALIDATION_SETTINGS</code>
Value: 0
Indicates the parameter is type D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS .

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS structure (d3d12sdklayers.h)

Article 02/22/2024

Describes settings used by GPU-Based Validation.

Syntax

C++

```
typedef struct D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS {
    UINT
    MaxMessagesPerCommandList;
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE
    DefaultShaderPatchMode;
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS
    PipelineStateCreateFlags;
} D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS;
```

Members

`MaxMessagesPerCommandList`

Specifies a `UINT` that limits the number of messages that can be stored in the GPU-Based Validation message log. The default value is 256. Since many identical errors can be produced in a single Draw/Dispatch call it may be useful to increase this number. Note this can become a memory burden if a large number of command lists are used as there is a committed message log per command list.

`DefaultShaderPatchMode`

Specifies the `D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE` that GPU-Based Validation uses when injecting validation code into shaders, except when overridden by per-command-list GPU-Based Validation settings (see `D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS`). The default value is

`D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_UNGUARDED_VALIDATION`.

`PipelineStateCreateFlags`

Specifies one of the [D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS](#) that indicates how GPU-Based Validation handles patching pipeline states. The default value is [D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_NONE](#).

Remarks

Point to an object using this structure with the *pData* member of [ID3D12DebugDevice1::SetDebugParameter](#) to configure device-wide GPU-Based Validation settings.

Individual command lists can override the default shader patch mode using [ID3D12DebugCommandList1::SetDebugParameter](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h (include D3d12sdklayers_RS1.h)

See also

[Debug Layer Structures](#)

[SetEnableGPUBasedValidation](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR structure (d3d12sdklayers.h)

Article04/02/2021

Describes the amount of artificial slowdown inserted by the debug device to simulate lower-performance graphics adapters.

Syntax

C++

```
typedef struct D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR {
    FLOAT SlowdownFactor;
} D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR;
```

Members

SlowdownFactor

Specifies the amount of slowdown artificially applied, as a factor of the nominal time for the fence to signal. The default value is 0.

Remarks

The SlowdownFactor is applied by artificially delaying the time it takes for a fence to signal. When SlowdownFactor is non-zero, the time taken for a fence to signal is approximately $1.0 + \text{SlowdownFactor}$ times the length of the nominal timing.

Requirements

 Expand table

Requirement	Value
Header	d3d12sdklayers.h (include D3D12.h)

See also

[Core Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_DEBUG_DEVICE_PARAMETER_TYPE enumeration (d3d12sdklayers.h)

Article02/22/2024

Specifies the data type of the memory pointed to by the *pData* parameter of [ID3D12DebugDevice1::SetDebugParameter](#) and [ID3D12DebugDevice1::GetDebugParameter](#).

Syntax

C++

```
typedef enum D3D12_DEBUG_DEVICE_PARAMETER_TYPE {
    D3D12_DEBUG_DEVICE_PARAMETER_FEATURE_FLAGS = 0,
    D3D12_DEBUG_DEVICE_PARAMETER_GPU_BASED_VALIDATION_SETTINGS,
    D3D12_DEBUG_DEVICE_PARAMETER_GPU_SLOWDOWN_PERFORMANCE_FACTOR
};
```

Constants

 Expand table

`D3D12_DEBUG_DEVICE_PARAMETER_FEATURE_FLAGS`

Value: 0

Indicates *pData* points to a [D3D12_DEBUG_FEATURE](#) value.

`D3D12_DEBUG_DEVICE_PARAMETER_GPU_BASED_VALIDATION_SETTINGS`

Indicates *pData* points to a [D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#) structure.

`D3D12_DEBUG_DEVICE_PARAMETER_GPU_SLOWDOWN_PERFORMANCE_FACTOR`

Indicates *pData* points to a [D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR](#) structure.

Requirements

 Expand table

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_DEBUG_FEATURE enumeration (d3d12sdklayers.h)

Article02/22/2024

Flags for optional D3D12 Debug Layer features.

Syntax

C++

```
typedef enum D3D12_DEBUG_FEATURE {
    D3D12_DEBUG_FEATURE_NONE = 0,
    D3D12_DEBUG_FEATURE_ALLOW_BEHAVIOR_CHANGING_DEBUG_AIDS = 0x1,
    D3D12_DEBUG_FEATURE_CONSERVATIVE_RESOURCE_STATE_TRACKING = 0x2,
    D3D12_DEBUG_FEATURE_DISABLE_VIRTUALIZED_BUNDLES_VALIDATION = 0x4,
    D3D12_DEBUG_FEATURE_EMULATE_WINDOWST7
} ;
```

Constants

[+] Expand table

D3D12_DEBUG_FEATURE_NONE

Value: 0

The default. No optional Debug Layer features.

D3D12_DEBUG_FEATURE_ALLOW_BEHAVIOR_CHANGING_DEBUG_AIDS

Value: 0x1

The Debug Layer is allowed to deliberately change functional behavior of an application in order to help identify potential errors. By default, the Debug Layer allows most invalid API usage to run the natural course.

D3D12_DEBUG_FEATURE_CONSERVATIVE_RESOURCE_STATE_TRACKING

Value: 0x2

Performs additional resource state validation of resources set in descriptors at the time [ID3D12CommandQueue::ExecuteCommandLists](#) is called. By design descriptors can be changed even after submitting command lists assuming proper synchronization. Conservative resource state tracking ignores this allowance and validates all resources used in descriptor tables when [ExecuteCommandLists](#) is called. The result may be false validation errors.

D3D12_DEBUG_FEATURE_DISABLE_VIRTUALIZED_BUNDLES_VALIDATION

Value: 0x4

Disables validation of bundle commands by virtually injecting checks into the calling command list validation paths.

Remarks

This enum is used by [ID3D12DebugDevice1::SetDebugParameter](#) and [ID3D12DebugDevice1::GetDebugParameter](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GPU_BASED_VALIDATION_FLAGS enumeration (d3d12sdklayers.h)

Article02/22/2024

Describes the level of GPU-based validation to perform at runtime.

Syntax

C++

```
typedef enum D3D12_GPU_BASED_VALIDATION_FLAGS {
    D3D12_GPU_BASED_VALIDATION_FLAGS_NONE = 0,
    D3D12_GPU_BASED_VALIDATION_FLAGS_DISABLE_STATE_TRACKING = 0x1
};
```

Constants

[+] Expand table

D3D12_GPU_BASED_VALIDATION_FLAGS_NONE

Value: 0

Default behavior; resource states, descriptors, and descriptor tables are all validated.

D3D12_GPU_BASED_VALIDATION_FLAGS_DISABLE_STATE_TRACKING

Value: 0x1

When set, GPU-based validation does not perform resource state validation which greatly reduces the performance cost of GPU-based validation. Descriptors and descriptor heaps are still validated.

Remarks

This enumeration is used with the [ID3D12Debug2::SetGPUValidationFlags](#) method to configure the amount of runtime validation that will occur.

Requirements

[+] Expand table

Requirement	Value
Header	d3d12sdklayers.h

See also

[Core Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS enumeration (d3d12sdklayers.h)

Article01/31/2022

Specifies how GPU-Based Validation handles patched pipeline states during [ID3D12Device::CreateGraphicsPipelineState](#) and [ID3D12Device::CreateComputePipelineState](#).

Syntax

C++

```
typedef enum D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS {
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_NONE = 0,
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_TRACKING_ONLY_SHADERS = 0x1,
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_UNGUARDED_VALIDATION_SHADERS = 0x2,
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_GUARDED_VALIDATION_SHADERS = 0x4,
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS_VALID_MASK = 0x7
};
```

Constants

[] [Expand table](#)

<code>D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_NONE</code>

Value: *0*

This is the default value. Indicates no patching of pipeline states should be done during PSO creation. Instead PSO's are patched on first use in a command list. This can help to reduce the up-front cost of PSO creation but may instead slow down command list recording until a steady-state is reached.

<code>D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_TRACKING_ONLY_SHADERS</code>
--

Value: *0x1*

Indicates that state-tracking GPU-Based Validation PSO's should be created along with the original PSO at create time.
<code>D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_UNGUARDED_VALIDATION_SHADERS</code> Value: <i>0x2</i> Indicates that unguarded GPU-Based Validation PSO's should be created along with the original PSO at create time.
<code>D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_GUARDED_VALIDATION_SHADERS</code> Value: <i>0x4</i> Indicates that guarded GPU-Based Validation PSO's should be created along with the original PSO at create time.
<code>D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS_VALID_MASK</code> Value: <i>0x7</i> Internal use only.

Remarks

This enum is used by the [D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#) structure.

Generally speaking most application developers are likely to leave this parameter unchanged. However, if the overhead of deferring patched PSO creation is suspected to be too much of a performance problem, then developers should consider changing this setting.

Requirements

[Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE enumeration (d3d12sdklayers.h)

Article02/22/2024

Specifies the type of shader patching used by GPU-Based Validation at either the device or command list level.

Syntax

C++

```
typedef enum D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE {
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_NONE = 0,
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_STATE_TRACKING_ONLY,
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_UNGUARDED_VALIDATION,
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_GUARDED_VALIDATION,
    NUM_D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODES
} ;
```

Constants

 Expand table

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_NONE

Value: 0

No shader patching is to be done. This will retain the original shader bytecode. Can lead to errors in some of the GPU-Based Validation state tracking as the unpatched shader may still change resource state (see [Common state promotion](#)) but the promotion will be untracked without patching the shader. This can improve performance but no validation will be performed and may also lead to misleading GPU-Based Validation errors. Use this mode very carefully.

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_STATE_TRACKING_ONLY

Shaders can be patched with resource state tracking code but no validation. This may improve performance but no validation will be performed.

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_UNGUARDED_VALIDATION

The default. Shaders are patched with validation code but erroneous instructions will still be executed.

`D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_GUARDED_VALIDATION`

Shaders are patched with validation code and erroneous instructions are skipped in execution. This can help avoid crashes or device removal.

`NUM_D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODES`

Unused, simply the count of the number of modes.

Remarks

This enum is used by the [D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS](#) structure.

Requirements

[+] [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INFO_QUEUE_FILTER structure (d3d12sdklayers.h)

Article02/22/2024

Debug message filter; contains a lists of message types to allow or deny.

Syntax

C++

```
typedef struct D3D12_INFO_QUEUE_FILTER {
    D3D12_INFO_QUEUE_FILTER_DESC AllowList;
    D3D12_INFO_QUEUE_FILTER_DESC DenyList;
} D3D12_INFO_QUEUE_FILTER;
```

Members

AllowList

Specifies types of messages that you want to allow. See [D3D12_INFO_QUEUE_FILTER_DESC](#).

DenyList

Specifies types of messages that you want to deny.

Remarks

For use with an [ID3D12InfoQueue](#) Interface.

Requirements

 Expand table

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_INFO_QUEUE_FILTER_DESC structure (d3d12sdklayers.h)

Article 07/27/2022

Allow or deny certain types of messages to pass through a filter.

Syntax

C++

```
typedef struct D3D12_INFO_QUEUE_FILTER_DESC {
    UINT NumCategories;
    D3D12_MESSAGE_CATEGORY *pCategoryList;
    UINT NumSeverities;
    D3D12_MESSAGE_SEVERITY *pSeverityList;
    UINT NumIDs;
    D3D12_MESSAGE_ID *pIDList;
} D3D12_INFO_QUEUE_FILTER_DESC;
```

Members

`NumCategories`

Number of message categories to allow or deny.

`pCategoryList`

Array of message categories to allow or deny. Array must have at least *NumCategories* members (see [D3D12_MESSAGE_CATEGORY](#)).

`NumSeverities`

Number of message severity levels to allow or deny.

`pSeverityList`

Array of message severity levels to allow or deny. Array must have at least *NumSeverities* members (see [D3D12_MESSAGE_SEVERITY](#)).

`NumIDs`

Number of message IDs to allow or deny.

pIDLList

Array of message IDs to allow or deny. Array must have at least *NumIDs* members (see [D3D12_MESSAGE_ID](#)).

Remarks

For use with an [ID3D12InfoQueue](#) Interface.

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Structures](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MESSAGE structure (d3d12sdklayers.h)

Article 02/22/2024

A debug message in the Information Queue.

Syntax

C++

```
typedef struct D3D12_MESSAGE {
    D3D12_MESSAGE_CATEGORY Category;
    D3D12_MESSAGE_SEVERITY Severity;
    D3D12_MESSAGE_ID       ID;
    const char              *pDescription;
    SIZE_T                  DescriptionByteLength;
} D3D12_MESSAGE;
```

Members

Category

The category of the message. See [D3D12_MESSAGE_CATEGORY](#).

Severity

The severity of the message. See [D3D12_MESSAGE_SEVERITY](#).

ID

The ID of the message. See [D3D12_MESSAGE_ID](#).

pDescription

The message string.

DescriptionByteLength

The length of *pDescription*, in bytes.

Remarks

This structure is returned from [ID3D12InfoQueue::GetMessage](#) as part of the Information Queue feature (see [ID3D12InfoQueue](#)).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MESSAGE_CATEGORY enumeration (d3d12sdklayers.h)

Article02/22/2024

Specifies categories of debug messages. This will identify the category of a message when retrieving a message with [ID3D12InfoQueue::GetMessage](#) and when adding a message with [ID3D12InfoQueue::AddMessage](#). When creating an info queue filter, these values can be used to allow or deny any categories of messages to pass through the storage and retrieval filters.

Syntax

C++

```
typedef enum D3D12_MESSAGE_CATEGORY {
    D3D12_MESSAGE_CATEGORY_APPLICATION_DEFINED = 0,
    D3D12_MESSAGE_CATEGORY_MISCELLANEOUS,
    D3D12_MESSAGE_CATEGORY_INITIALIZATION,
    D3D12_MESSAGE_CATEGORY_CLEANUP,
    D3D12_MESSAGE_CATEGORY_COMPILATION,
    D3D12_MESSAGE_CATEGORY_STATE_CREATION,
    D3D12_MESSAGE_CATEGORY_STATE_SETTING,
    D3D12_MESSAGE_CATEGORY_STATE_GETTING,
    D3D12_MESSAGE_CATEGORY_RESOURCE_MANIPULATION,
    D3D12_MESSAGE_CATEGORY_EXECUTION,
    D3D12_MESSAGE_CATEGORY_SHADER
};
```

Constants

[] Expand table

<code>D3D12_MESSAGE_CATEGORY_APPLICATION_DEFINED</code>
Value: 0
Indicates a user defined message, see ID3D12InfoQueue::AddMessage .
<code>D3D12_MESSAGE_CATEGORY_MISCELLANEOUS</code>
<code>D3D12_MESSAGE_CATEGORY_INITIALIZATION</code>
<code>D3D12_MESSAGE_CATEGORY_CLEANUP</code>

D3D12_MESSAGE_CATEGORY_COMPILATION
D3D12_MESSAGE_CATEGORY_STATE_CREATION
D3D12_MESSAGE_CATEGORY_STATE_SETTING
D3D12_MESSAGE_CATEGORY_STATE_GETTING
D3D12_MESSAGE_CATEGORY_RESOURCE_MANIPULATION
D3D12_MESSAGE_CATEGORY_EXECUTION
D3D12_MESSAGE_CATEGORY_SHADER

Remarks

This is part of the Information Queue feature, refer to the [ID3D12InfoQueue](#) Interface.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MESSAGE_ID enumeration (d3d12sdklayers.h)

Article02/14/2023

Specifies debug message IDs for setting up an info-queue filter (see [D3D12_INFO_QUEUE_FILTER](#)); use these messages to allow or deny message categories to pass through the storage and retrieval filters. These IDs are used by methods such as [ID3D12InfoQueue::GetMessage](#) or [ID3D12InfoQueue::AddMessage](#).

Syntax

C++

```
typedef enum D3D12_MESSAGE_ID {
    D3D12_MESSAGE_ID_UNKNOWN = 0,
    D3D12_MESSAGE_ID_STRING_FROM_APPLICATION = 1,
    D3D12_MESSAGE_ID_CORRUPTED_THIS = 2,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER1 = 3,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER2 = 4,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER3 = 5,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER4 = 6,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER5 = 7,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER6 = 8,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER7 = 9,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER8 = 10,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER9 = 11,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER10 = 12,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER11 = 13,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER12 = 14,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER13 = 15,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER14 = 16,
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER15 = 17,
    D3D12_MESSAGE_ID_CORRUPTED_MULTITHREADING = 18,
    D3D12_MESSAGE_ID_MESSAGE_REPORTING_OUTOFCMEMORY = 19,
    D3D12_MESSAGE_ID_GETPRIVATEDATA_MOREDATA = 20,
    D3D12_MESSAGE_ID_SETPRIVATEDATA_INVALIDFREEDATA = 21,
    D3D12_MESSAGE_ID_SETPRIVATEDATA_CHANGINGPARAMS = 24,
    D3D12_MESSAGE_ID_SETPRIVATEDATA_OUTOFCMEMORY = 25,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_UNRECOGNIZEDFORMAT = 26,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDESC = 27,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFORMAT = 28,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDVIDEOPLANESLICE = 29,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDPLANESLICE = 30,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDIMENSIONS = 31,
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDRESOURCE = 32,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNRECOGNIZEDFORMAT = 35,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNSUPPORTEDFORMAT = 36,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDESC = 37,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDFORMAT = 38,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDVIDEOPLANESLICE = 39,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDPLANESLICE = 40,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDIMENSIONS = 41,
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDRESOURCE = 42,
```

```
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_UNRECOGNIZEDFORMAT = 45,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_INVALIDDESC = 46,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_INVALIDFORMAT = 47,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_INVALIDDIMENSIONS = 48,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_INVALIDRESOURCE = 49,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_OUTOFGMEMORY = 52,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TOOMANYELEMENTS = 53,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDFORMAT = 54,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INCOMPATIBLEFORMAT = 55,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSLOT = 56,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDINPUTSLOTCLASS = 57,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_STEPSLOTCLASSMISMATCH = 58,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSLOTCLASSCHANGE = 59,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSTEPRATECHANGE = 60,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDALIGNMENT = 61,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_DUPLICATESEMANTIC = 62,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_UNPARSEABLEINPUTSIGNATURE = 63,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_NULLSEMANTIC = 64,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_MISSINGELEMENT = 65,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_OUTOFGMEMORY = 66,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERBYTECODE = 67,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERTYPE = 68,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_OUTOFGMEMORY = 69,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERBYTECODE = 70,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERTYPE = 71,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTOFGMEMORY = 72,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERBYTECODE = 73,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE = 74,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMENTRIES = 75,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSTREAMSTRIDEUNUSED =
76,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSLOT0EXPECTED = 79,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSLOT = 80,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_ONLYONEELEMENTPERSLOT = 81,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCOMPONENTCOUNT = 82,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTARTCOMPONENTANDCOMPONE
NTCOUNT = 83,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDGAPDEFINITION = 84,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_REPEATODOPUT = 85,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSTREAMSTRIDE =
86,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGSEMANTIC = 87,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MASKMISMATCH = 88,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_CANTHAVEONLYGAPS = 89,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DECLTOOCOMPLEX = 90,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGOUTPUTSIGNATURE = 91,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_OUTOFGMEMORY = 92,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERBYTECODE = 93,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERTYPE = 94,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFILLMODE = 95,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDCULLMODE = 96,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDDEPTHBIASCLAMP = 97,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDSLOPESCALEDEDEPTHBIAS = 98,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDDEPTHWRITEMASK = 100,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDDEPTHFUNC = 101,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDFRONTFACE_STENCILFAILOP = 102,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDFRONTFACE_STENCILZFAILOP = 103,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDFRONTFACE_STENCILPASSOP = 104,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_INVALIDFRONTFACE_STENCILFUNC = 105,
```

```
D3D12_MESSAGE_ID_CREATEDDEPTH_STENCILSTATE_INVALIDBACKFACE_STENCILFAILOP = 106,
D3D12_MESSAGE_ID_CREATEDDEPTH_STENCILSTATE_INVALIDBACKFACE_STENCILZFAILOP = 107,
D3D12_MESSAGE_ID_CREATEDDEPTH_STENCILSTATE_INVALIDBACKFACE_STENCILPASSOP = 108,
D3D12_MESSAGE_ID_CREATEDDEPTH_STENCILSTATE_INVALIDBACKFACE_STENCILFUNC = 109,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND = 111,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEND = 112,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP = 113,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLENDALPHA = 114,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLENDALPHA = 115,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOPALPHA = 116,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDRENDERTARGETWRITEMASK = 117,
D3D12_MESSAGE_ID_GET_PROGRAM_IDENTIFIER_ERROR,
D3D12_MESSAGE_ID_GET_WORK_GRAPH_PROPERTIES_ERROR,
D3D12_MESSAGE_ID_SET_PROGRAM_ERROR,
D3D12_MESSAGE_ID_CLEARDEPTH_STENCILVIEW_INVALID = 135,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_NOT_SET = 200,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_MISMATCH = 201,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_NOT_SET = 202,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_STRIDE_TOO_SMALL = 209,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_TOO_SMALL = 210,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_NOT_SET = 211,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_FORMAT_INVALID = 212,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_TOO_SMALL = 213,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INVALID_PRIMITIVE_TOPOLOGY = 219,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_STRIDE_UNALIGNED = 221,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_OFFSET_UNALIGNED = 222,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_AT_FAULT = 232,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_POSSIBLY_AT_FAULT = 233,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_NOT_AT_FAULT = 234,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TRAILING_DIGIT_IN_SEMANTIC = 239,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_TRAILING_DIGIT_IN_SEMANTIC =
240,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TYPE_MISMATCH = 245,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_EMPTY_LAYOUT = 253,
D3D12_MESSAGE_ID_LIVE_OBJECT_SUMMARY = 255,
D3D12_MESSAGE_ID_LIVE_DEVICE = 274,
D3D12_MESSAGE_ID_LIVE_SWAPCHAIN = 275,
D3D12_MESSAGE_ID_CREATEDDEPTH_STENCILVIEW_INVALIDFLAGS = 276,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDCLASSLINKAGE = 277,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDCLASSLINKAGE = 278,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAMTORASTERIZER =
280,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDCLASSLINKAGE = 283,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAM = 284,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDENTRIES = 285,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDSTRIDES = 286,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMSTRIDES = 287,
D3D12_MESSAGE_ID_CREATEHULLSHADER_OUTOFMEMORY = 289,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERBYTECODE = 290,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERTYPE = 291,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDCLASSLINKAGE = 292,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_OUTOFMEMORY = 294,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERBYTECODE = 295,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERTYPE = 296,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCLASSLINKAGE = 297,
D3D12_MESSAGE_ID_RESOURCE_UNMAP_NOTMAPPED = 310,
D3D12_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_MISMATCHED_DATA_SIZE = 318,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_OUTOFMEMORY = 321,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERBYTECODE = 322,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCLASSLINKAGE = 323,
```

```
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEFLOATOPSNOTSUPPORTED = 331,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEFLOATOPSNOTSUPPORTED = 332,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEFLOATOPSNOTSUPPORTED = 333,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEFLOATOPSNOTSUPPORTED = 334,

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEFLOATOPSNOTSUPPORT
ED = 335,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEFLOATOPSNOTSUPPORTED = 336,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEFLOATOPSNOTSUPPORTED = 337,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDRESOURCE = 340,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDESC = 341,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFORMAT = 342,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDVIDEOPLANESLICE = 343,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDPLANESLICE = 344,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDIMENSIONS = 345,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_UNRECOGNIZEDFORMAT = 346,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFLAGS = 354,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFORCEDSAMPLECOUNT = 401,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDLOGICOPS = 403,
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 410,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 412,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 414,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 416,

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEEXTENSIONSNOTSUPPO
RTED = 418,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 420,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEEXTENSIONSNOTSUPPORTED = 422,
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_UAVSNOTSUPPORTED = 425,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_UAVSNOTSUPPORTED = 426,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_UAVSNOTSUPPORTED = 427,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_UAVSNOTSUPPORTED = 428,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UAVSNOTSUPPORTED =
429,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_UAVSNOTSUPPORTED = 430,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_UAVSNOTSUPPORTED = 431,
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_INVALIDSOURCERECT = 447,
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_EMPTYRECT = 448,
D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_INVALID_PARAMETER = 493,
D3D12_MESSAGE_ID_COPYTILEMAPPINGS_INVALID_PARAMETER = 494,
D3D12_MESSAGE_ID_CREATEDevice_INVALIDARGS = 506,
D3D12_MESSAGE_ID_CREATEDevice_WARNING = 507,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_TYPE = 519,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_NULL_POINTER = 520,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SUBRESOURCE = 521,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_RESERVED_BITS = 522,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISSING_BIND_FLAGS = 523,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_MISC_FLAGS = 524,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MATCHING_STATES = 525,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINATION = 526,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_BEFORE_AFTER_MISMATCH = 527,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_RESOURCE = 528,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_SAMPLE_COUNT = 529,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS = 530,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINED_FLAGS = 531,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS_FOR_FORMAT = 532,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SPLIT_BARRIER = 533,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_END = 534,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_BEGIN = 535,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAG = 536,
```

```
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMMAND_LIST_TYPE = 537,
D3D12_MESSAGE_ID_INVALID_SUBRESOURCE_STATE = 538,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CONTENTION = 540,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET = 541,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET_BUNDLE = 542,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CANNOT_RESET = 543,
D3D12_MESSAGE_ID_COMMAND_LIST_OPEN = 544,
D3D12_MESSAGE_ID_INVALID_BUNDLE_API = 546,
D3D12_MESSAGE_ID_COMMAND_LIST_CLOSED = 547,
D3D12_MESSAGE_ID_WRONG_COMMAND_ALLOCATOR_TYPE = 549,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_SYNC = 552,
D3D12_MESSAGE_ID_COMMAND_LIST_SYNC = 553,
D3D12_MESSAGE_ID_SET_DESCRIPTOR_HEAP_INVALID = 554,
D3D12_MESSAGE_ID_CREATE_COMMANDQUEUE = 557,
D3D12_MESSAGE_ID_CREATE_COMMANDALLOCATOR = 558,
D3D12_MESSAGE_ID_CREATE_PIPELINESTATE = 559,
D3D12_MESSAGE_ID_CREATE_COMMANDLIST12 = 560,
D3D12_MESSAGE_ID_CREATE_RESOURCE = 562,
D3D12_MESSAGE_ID_CREATE_DESCRIPTORHEAP = 563,
D3D12_MESSAGE_ID_CREATE_ROOTSIGNATURE = 564,
D3D12_MESSAGE_ID_CREATE_LIBRARY = 565,
D3D12_MESSAGE_ID_CREATE_HEAP = 566,
D3D12_MESSAGE_ID_CREATE_MONITOREDFENCE = 567,
D3D12_MESSAGE_ID_CREATE_QUERYHEAP = 568,
D3D12_MESSAGE_ID_CREATE_COMMANDSIGNATURE = 569,
D3D12_MESSAGE_ID_LIVE_COMMANDQUEUE = 570,
D3D12_MESSAGE_ID_LIVE_COMMANDALLOCATOR = 571,
D3D12_MESSAGE_ID_LIVE_PIPELINESTATE = 572,
D3D12_MESSAGE_ID_LIVE_COMMANDLIST12 = 573,
D3D12_MESSAGE_ID_LIVE_RESOURCE = 575,
D3D12_MESSAGE_ID_LIVE_DESCRIPTORHEAP = 576,
D3D12_MESSAGE_ID_LIVE_ROOTSIGNATURE = 577,
D3D12_MESSAGE_ID_LIVE_LIBRARY = 578,
D3D12_MESSAGE_ID_LIVE_HEAP = 579,
D3D12_MESSAGE_ID_LIVE_MONITOREDFENCE = 580,
D3D12_MESSAGE_ID_LIVE_QUERYHEAP = 581,
D3D12_MESSAGE_ID_LIVE_COMMANDSIGNATURE = 582,
D3D12_MESSAGE_ID_DESTROY_COMMANDQUEUE = 583,
D3D12_MESSAGE_ID_DESTROY_COMMANDALLOCATOR = 584,
D3D12_MESSAGE_ID_DESTROY_PIPELINESTATE = 585,
D3D12_MESSAGE_ID_DESTROY_COMMANDLIST12 = 586,
D3D12_MESSAGE_ID_DESTROY_RESOURCE = 588,
D3D12_MESSAGE_ID_DESTROY_DESCRIPTORHEAP = 589,
D3D12_MESSAGE_ID_DESTROY_ROOTSIGNATURE = 590,
D3D12_MESSAGE_ID_DESTROY_LIBRARY = 591,
D3D12_MESSAGE_ID_DESTROY_HEAP = 592,
D3D12_MESSAGE_ID_DESTROY_MONITOREDFENCE = 593,
D3D12_MESSAGE_ID_DESTROY_QUERYHEAP = 594,
D3D12_MESSAGE_ID_DESTROY_COMMANDSIGNATURE = 595,
D3D12_MESSAGE_ID_CREATEROFILE_INVALIDDIMENSIONS = 597,
D3D12_MESSAGE_ID_CREATEROFILE_INVALIDMISCFLAGS = 599,
D3D12_MESSAGE_ID_CREATEROFILE_INVALIDARG_RETURN = 602,
D3D12_MESSAGE_ID_CREATEROFILE_OUTOFMEMORY_RETURN = 603,
D3D12_MESSAGE_ID_CREATEROFILE_INVALIDDESC = 604,
D3D12_MESSAGE_ID_POSSIBLY_INVALID_SUBRESOURCE_STATE = 607,
D3D12_MESSAGE_ID_INVALID_USE_OF_NON_RESIDENT_RESOURCE = 608,
D3D12_MESSAGE_ID_POSSIBLE_INVALID_USE_OF_NON_RESIDENT_RESOURCE = 609,
D3D12_MESSAGE_ID_BUNDLE_PIPELINE_STATE_MISMATCH = 610,
D3D12_MESSAGE_ID_PRIMITIVE_TOPOLOGY_MISMATCH_PIPELINE_STATE = 611,
D3D12_MESSAGE_ID_RENDER_TARGET_FORMAT_MISMATCH_PIPELINE_STATE = 613,
```

```
D3D12_MESSAGE_ID_RENDER_TARGET_SAMPLE_DESC_MISMATCH_PIPELINE_STATE = 614,
D3D12_MESSAGE_ID_DEPTH_STENCIL_FORMAT_MISMATCH_PIPELINE_STATE = 615,
D3D12_MESSAGE_ID_DEPTH_STENCIL_SAMPLE_DESC_MISMATCH_PIPELINE_STATE = 616,
D3D12_MESSAGE_ID_CREATESHADER_INVALIDBYTECODE = 622,
D3D12_MESSAGE_ID_CREATEHEAP_NULLDESC = 623,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDSIZE = 624,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDHEAPTYPE = 625,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES = 626,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMEMORYPOOL = 627,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDPROPERTIES = 628,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDALIGNMENT = 629,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMISCFLAGS = 630,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDMISCFLAGS = 631,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDARG_RETURN = 632,
D3D12_MESSAGE_ID_CREATEHEAP_OUTOFGMEMORY_RETURN = 633,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLHEAPPROPERTIES = 634,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPTYPE = 635,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES = 636,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDMEMORYPOOL = 637,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPPROPERTIES = 638,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPMISCFLAGS = 639,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPMISCFLAGS = 640,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDARG_RETURN = 641,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_OUTOFGMEMORY_RETURN = 642,
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_UNRECOGNIZEDHEAPTYPE = 643,
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_INVALIDHEAPTYPE = 644,
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_INVALID_DESC = 645,
D3D12_MESSAGE_ID_INVALID_DESCRIPTOR_HANDLE = 646,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALID_CONSERVATIVERASTERMODE = 647,
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_RESOURCE = 649,
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_DESC = 650,
D3D12_MESSAGE_ID_CREATE_UNORDEREDACCESS_VIEW_INVALID_COUNTER_USAGE = 652,
D3D12_MESSAGE_ID_COPY_DESCRIPTORSS_INVALID_RANGES = 653,
D3D12_MESSAGE_ID_COPY_DESCRIPTORSS_WRITE_ONLY_DESCRIPTOR = 654,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_RTV_FORMAT_NOT_UNKNOWN = 655,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_RENDER_TARGET_COUNT = 656,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_VERTEX_SHADER_NOT_SET = 657,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INPUTLAYOUT_NOT_SET = 658,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_HS_DS_SIGNATURE_MISMATCH
= 659,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_REGISTERINDEX = 660,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_COMPONENTTYPE = 661,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_REGISTERMASK = 662,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_SYSTEMVALUE = 663,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_NEVERWRITTEN_ALWAYSREADS
= 664,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_MINPRECISION = 665,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_SEMANTICNAME_NOT_FOUND
= 666,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_XOR_DS_MISMATCH = 667,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HULL_SHADER_INPUT_TOPOLOGY_MISMATCH =
668,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_DS_CONTROL_POINT_COUNT_MISMATCH =
669,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_DS_TESSELLATOR_DOMAIN_MISMATCH =
670,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_USE_OF_CENTER_MULTISAMPLE_PATTER
```

```
N = 671,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_INVALID_USE_OF_FORCED_SAMPLE_COUNT =
672,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_INVALID_PRIMITIVETOPOLOGY = 673,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_INVALID_SYSTEMVALUE = 674,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_OM_DUAL_SOURCE_BLENDING_CAN_ONLY_HAVE_RENDER_TARGET_0 = 675,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_OM_RENDER_TARGET_DOES_NOT_SUPPORT_BLENDING = 676,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_PS_OUTPUT_TYPE_MISMATCH = 677,

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_OM_RENDER_TARGET_DOES_NOT_SUPPORT_LOGIC_OPS = 678,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_RenderTargetView_Not_Set = 679,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_DepthStencilView_Not_Set = 680,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_GS_Input_Primitive_Mismatch = 681,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Position_Not_Present = 682,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Missing_Root_Signature_Flags = 683,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Index_Buffer_Properties = 684,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Sample_Desc = 685,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_HS_Root_Signature_Mismatch = 686,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_DS_Root_Signature_Mismatch = 687,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_VS_Root_Signature_Mismatch = 688,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_GS_Root_Signature_Mismatch = 689,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_PS_Root_Signature_Mismatch = 690,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Missing_Root_Signature = 691,
D3D12_MESSAGE_ID_Execute_Bundle_Open_Bundle = 692,
D3D12_MESSAGE_ID_Execute_Bundle_Descriptor_Heap_Mismatch = 693,
D3D12_MESSAGE_ID_Execute_Bundle_Type = 694,
D3D12_MESSAGE_ID_Draw_Empty_Scissor_Rectangle = 695,
D3D12_MESSAGE_ID_Create_Root_Signature_Blob_Not_Found = 696,
D3D12_MESSAGE_ID_Create_Root_Signature_Deserialize_Failed = 697,
D3D12_MESSAGE_ID_Create_Root_Signature_Invalid_Configuration = 698,
D3D12_MESSAGE_ID_Create_Root_Signature_Not_Supported_On_Device = 699,
D3D12_MESSAGE_ID_Createresourceandheap_NullResourceProperties = 700,
D3D12_MESSAGE_ID_Createresourceandheap_NullHeap = 701,
D3D12_MESSAGE_ID_GetResourceAllocationInfo_InvalidRdescs = 702,
D3D12_MESSAGE_ID_Makeresident_Nullobjectarray = 703,
D3D12_MESSAGE_ID_Evict_Nullobjectarray = 705,
D3D12_MESSAGE_ID_Set_Descriptor_Table_Invalid = 708,
D3D12_MESSAGE_ID_Set_Root_Constant_Invalid = 709,
D3D12_MESSAGE_ID_Set_Root_Constant_Buffer_View_Invalid = 710,
D3D12_MESSAGE_ID_Set_Root_Shader_Resource_View_Invalid = 711,
D3D12_MESSAGE_ID_Set_Root_Unordered_Access_View_Invalid = 712,
D3D12_MESSAGE_ID_Set_Vertex_Buffers_Invalid_Desc = 713,
D3D12_MESSAGE_ID_Set_Index_Buffer_Invalid_Desc = 715,
D3D12_MESSAGE_ID_Set_Stream_Output_Buffers_Invalid_Desc = 717,
D3D12_MESSAGE_ID_Createresource_UnrecognizedDimensionality = 718,
D3D12_MESSAGE_ID_Createresource_UnrecognizedLayout = 719,
D3D12_MESSAGE_ID_Createresource_InvalidDimensionality = 720,
D3D12_MESSAGE_ID_Createresource_InvalidAlignment = 721,
D3D12_MESSAGE_ID_Createresource_InvalidMiplevels = 722,
D3D12_MESSAGE_ID_Createresource_InvalidSampledesc = 723,
D3D12_MESSAGE_ID_Createresource_InvalidLayout = 724,
D3D12_MESSAGE_ID_Set_Index_Buffer_Invalid = 725,
D3D12_MESSAGE_ID_Set_Vertex_Buffers_Invalid = 726,
D3D12_MESSAGE_ID_Set_Stream_Output_Buffers_Invalid = 727,
D3D12_MESSAGE_ID_Set_Render_Targets_Invalid = 728,
```

```
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_INVALID_PARAMETERS = 729,
D3D12_MESSAGE_ID_BEGIN_END_QUERY_INVALID_PARAMETERS = 731,
D3D12_MESSAGE_ID_CLOSE_COMMAND_LIST_OPEN_QUERY = 732,
D3D12_MESSAGE_ID_RESOLVE_QUERY_DATA_INVALID_PARAMETERS = 733,
D3D12_MESSAGE_ID_SET_PREDICATION_INVALID_PARAMETERS = 734,
D3D12_MESSAGE_ID_TIMESTAMP_NOT_SUPPORTED = 735,
D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDFORMAT = 737,
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDFORMAT = 738,
D3D12_MESSAGE_ID_GETCOPYABLEFOOTPRINTS_INVALIDSUBRESOURCERANGE = 739,
D3D12_MESSAGE_ID_GETCOPYABLEFOOTPRINTS_INVALIDBASEOFFSET = 740,
D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDSUBRESOURCERANGE = 739,
D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDBASEOFFSET = 740,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_HEAP = 741,
D3D12_MESSAGE_ID_CREATE_SAMPLER_INVALID = 742,
D3D12_MESSAGE_ID_CREATECOMMANDSIGNATURE_INVALID = 743,
D3D12_MESSAGE_ID_EXECUTE_INDIRECT_INVALID_PARAMETERS = 744,
D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_RESOURCE_DIMENSION = 745,
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDCLEARVALUE = 815,
D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDCLEARVALUEFORMAT = 816,
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDCLEARVALUEFORMAT = 817,
D3D12_MESSAGE_ID_CREATERESOURCE_CLEARVALUEDENORMFLUSH = 818,
D3D12_MESSAGE_ID_CLEARRENDERTARGETVIEW_MISMATCHINGCLEARVALUE = 820,
D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_MISMATCHINGCLEARVALUE = 821,
D3D12_MESSAGE_ID_MAP_INVALIDHEAP = 822,
D3D12_MESSAGE_ID_UNMAP_INVALIDHEAP = 823,
D3D12_MESSAGE_ID_MAP_INVALIDRESOURCE = 824,
D3D12_MESSAGE_ID_UNMAP_INVALIDRESOURCE = 825,
D3D12_MESSAGE_ID_MAP_INVALIDSUBRESOURCE = 826,
D3D12_MESSAGE_ID_UNMAP_INVALIDSUBRESOURCE = 827,
D3D12_MESSAGE_ID_MAP_INVALIDRANGE = 828,
D3D12_MESSAGE_ID_UNMAP_INVALIDRANGE = 829,
D3D12_MESSAGE_ID_MAP_INVALIDDATAPORTER = 832,
D3D12_MESSAGE_ID_MAP_INVALIDARG_RETURN = 833,
D3D12_MESSAGE_ID_MAP_OUTOFMEMORY_RETURN = 834,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_BUNDLENOTSUPPORTED = 835,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_COMMANDLISTMISMATCH = 836,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_OPENCOMMANDLIST = 837,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_FAILEDCOMMANDLIST = 838,
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLDST = 839,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDDSTRESOURCEDIMENSION = 840,
D3D12_MESSAGE_ID_COPYBUFFERREGION_DSTRANGEOUTOFCOMMON = 841,
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLSRC = 842,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDSRCRESOURCEDIMENSION = 843,
D3D12_MESSAGE_ID_COPYBUFFERREGION_SRCRANGEOUTOFCOMMON = 844,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDCOPYFLAGS = 845,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLDST = 846,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDSTTYPE = 847,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRESOURCEDIMENSION = 848,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRESOURCE = 849,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTSUBRESOURCE = 850,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTOFFSET = 851,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDSTFORMAT = 852,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTFORMAT = 853,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDIMENSIONS = 854,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTROWPITCH = 855,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTPLACEMENT = 856,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDPLACEDFOOTPRINTFORMAT = 857,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_DSTREGIONOUTOFCOMMON = 858,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLSRC = 859,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCTYPE = 860,
```

```
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCEDIMENSION = 861,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCE = 862,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCERESOURCE = 863,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCOFFSET = 864,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCFORMAT = 865,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCFORMAT = 866,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDIMENSIONS = 867,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCROWPITCH = 868,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCPLACEMENT = 869,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDSPLACEDFOOTPRINTFORMAT = 870,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_SRCREGIONOUTOFCOMMANDLIST = 871,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTCOORDINATES = 872,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCBOX = 873,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_FORMATMISMATCH = 874,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_EMPTYBOX = 875,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDCOPYFLAGS = 876,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SUBRESOURCE_INDEX = 877,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_FORMAT = 878,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_MISMATCH = 879,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SAMPLE_COUNT = 880,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_INVALID_SHADER = 881,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_CS_ROOT_SIGNATURE_MISMATCH = 882,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_MISSING_ROOT_SIGNATURE = 883,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALIDCACHEDBLOB = 884,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBADAPTERMISMATCH = 885,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDRIVERVERSIONMISMATCH = 886,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDESCMISMATCH = 887,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBIGNORED = 888,
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDHEAP = 889,
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDRESOURCE = 890,
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDBOX = 891,
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDSUBRESOURCE = 892,
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_EMPTYBOX = 893,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDHEAP = 894,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDRESOURCE = 895,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDBOX = 896,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDSUBRESOURCE = 897,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_EMPTYBOX = 898,
D3D12_MESSAGE_ID_TOOMANYNODESSPECIFIED = 899,
D3D12_MESSAGE_ID_INVALID_NODE_INDEX = 900,
D3D12_MESSAGE_ID_GETTHEAPPROPERTIES_INVALIDRESOURCE = 901,
D3D12_MESSAGE_ID_NODE_MASK_MISMATCH = 902,
D3D12_MESSAGE_ID_COMMAND_LIST_OUTOFMEMORY = 903,
D3D12_MESSAGE_ID_COMMAND_LIST_MULTIPLE_SWAPCHAIN_BUFFER_REFERENCES = 904,
D3D12_MESSAGE_ID_COMMAND_LIST_TOOMANY_SWAPCHAIN_REFERENCES = 905,
D3D12_MESSAGE_ID_COMMAND_QUEUE_TOOMANY_SWAPCHAIN_REFERENCES = 906,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_WRONGSWAPCHAINBUFFERREFERENCE = 907,
D3D12_MESSAGE_ID_COMMAND_LIST_SETRENDERTARGETS_INVALIDNUMRENDERTARGETS = 908,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_TYPE = 909,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_FLAGS = 910,
D3D12_MESSAGE_ID_CREATESHAREDRSOURCE_INVALIDFLAGS = 911,
D3D12_MESSAGE_ID_CREATESHAREDRSOURCE_INVALIDFORMAT = 912,
D3D12_MESSAGE_ID_CREATESHAREDHEAP_INVALIDFLAGS = 913,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_UNRECOGNIZEDPROPERTIES = 914,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDSIZE = 915,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDOBJECT = 916,
D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDOBJECT = 917,
D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDKEY = 918,
D3D12_MESSAGE_ID_KEYEDMUTEX_WRONGSTATE = 919,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_PRIORITY = 920,
```

```
D3D12_MESSAGE_ID_OBJECT_DELETED_WHILE_STILL_IN_USE = 921,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_FLAGS = 922,
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_HAS_NO_RESOURCE = 923,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_RENDER_TARGET_DELETED = 924,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_ALL_RENDER_TARGETS_HAVE_UNKNOWN_FORMAT
= 925,
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_INTERSECTS_MULTIPLE_BUFFERS = 926,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_GPU_WRITTEN_READBACK_RESOURCE_MAPPED = 927,
D3D12_MESSAGE_ID_UNMAP_RANGE_NOT_EMPTY = 929,
D3D12_MESSAGE_ID_MAP_INVALID_NULLRANGE = 930,
D3D12_MESSAGE_ID_UNMAP_INVALID_NULLRANGE = 931,
D3D12_MESSAGE_ID_NO_GRAPHICS_API_SUPPORT = 932,
D3D12_MESSAGE_ID_NO_COMPUTE_API_SUPPORT = 933,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_FLAGS_NOT_SUPPORTED = 934,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_ARGUMENT_UNINITIALIZED = 935,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_HEAP_INDEX_OUT_OF_BOUNDS = 936,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TABLE_REGISTER_INDEX_OUT_OF_BOUNDS
= 937,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_UNINITIALIZED = 938,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TYPE_MISMATCH = 939,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SRV_RESOURCE_DIMENSION_MISMATCH = 940,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UAV_RESOURCE_DIMENSION_MISMATCH = 941,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INCOMPATIBLE_RESOURCE_STATE = 942,
D3D12_MESSAGE_ID_COPYRESOURCE_NULLDST = 943,
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDDSTRESOURCE = 944,
D3D12_MESSAGE_ID_COPYRESOURCE_NULLSRC = 945,
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDSRCRESOURCE = 946,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLDST = 947,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDDSTRESOURCE = 948,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLSRC = 949,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDSRCRESOURCE = 950,
D3D12_MESSAGE_ID_PIPELINE_STATE_TYPE_MISMATCH = 951,
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_NOT_SET = 952,
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_MISMATCH = 953,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_ZERO_BARRIERS = 954,
D3D12_MESSAGE_ID_BEGIN_END_EVENT_MISMATCH = 955,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_POSSIBLE_BEFORE_AFTER_MISMATCH = 956,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_BEGIN_END = 957,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INVALID_RESOURCE = 958,
D3D12_MESSAGE_ID_USE_OF_ZERO_REFCOUNT_OBJECT = 959,
D3D12_MESSAGE_ID_OBJECT_EVICTED_WHILE_STILL_IN_USE = 960,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_DESCRIPTOR_ACCESS_OUT_OF_BOUNDS = 961,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_INVALIDLIBRARYBLOB = 962,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_DRIVERVERSIONMISMATCH = 963,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_ADAPTERVERSIONMISMATCH = 964,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_UNSUPPORTED = 965,
D3D12_MESSAGE_ID_CREATE_PIPELINELIBRARY = 966,
D3D12_MESSAGE_ID_LIVE_PIPELINELIBRARY = 967,
D3D12_MESSAGE_ID_DESTROY_PIPELINELIBRARY = 968,
D3D12_MESSAGE_ID_STOREPIPELINE_NONAME = 969,
D3D12_MESSAGE_ID_STOREPIPELINE_DUPLICATENAME = 970,
D3D12_MESSAGE_ID_LOADPIPELINE_NAMENOTFOUND = 971,
D3D12_MESSAGE_ID_LOADPIPELINE_INVALIDDESC = 972,
D3D12_MESSAGE_ID_PIPELINELIBRARY_SERIALIZE_NOTENOUGHMEMORY = 973,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_PS_OUTPUT_RT_OUTPUT_MISMATCH = 974,
D3D12_MESSAGE_ID_SETEVENTONMULTIPLEFENCECOMPLETION_INVALIDFLAGS = 975,
D3D12_MESSAGE_ID_CREATE_QUEUE_VIDEO_NOT_SUPPORTED = 976,
D3D12_MESSAGE_ID_CREATE_COMMAND_ALLOCATOR_VIDEO_NOT_SUPPORTED = 977,
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_VIDEO_DECODE_STATISTICS_NOT_SUPPORTED = 978,
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDLIST = 979,
```

```
D3D12_MESSAGE_ID_CREATE_VIDEODECODER = 980,
D3D12_MESSAGE_ID_CREATE_VIDEODECODESTREAM = 981,
D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDLIST = 982,
D3D12_MESSAGE_ID_LIVE_VIDEODECODER = 983,
D3D12_MESSAGE_ID_LIVE_VIDEODECODESTREAM = 984,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDLIST = 985,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODER = 986,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODESTREAM = 987,
D3D12_MESSAGE_ID_DECODE_FRAME_INVALID_PARAMETERS = 988,
D3D12_MESSAGE_ID_DEPRECATED_API = 989,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_COMMAND_LIST_TYPE = 990,
D3D12_MESSAGE_ID_COMMAND_LIST_DESCRIPTOR_TABLE_NOT_SET = 991,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_CONSTANT_BUFFER_VIEW_NOT_SET = 992,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_SHADER_RESOURCE_VIEW_NOT_SET = 993,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_UNORDERED_ACCESS_VIEW_NOT_SET = 994,
D3D12_MESSAGE_ID_DISCARD_INVALID_SUBRESOURCE_RANGE = 995,
D3D12_MESSAGE_ID_DISCARD_ONE_SUBRESOURCE_FOR_MIPS_WITH_RECTS = 996,
D3D12_MESSAGE_ID_DISCARD_NO_RECTS_FOR_NON_TEXTURE2D = 997,
D3D12_MESSAGE_ID_COPY_ON_SAME_SUBRESOURCE = 998,
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PAGEABLE = 999,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UNSUPPORTED = 1000,
D3D12_MESSAGE_ID_STATIC_DESCRIPTOR_INVALID_DESCRIPTOR_CHANGE = 1001,
D3D12_MESSAGE_ID_DATA_STATIC_DESCRIPTOR_INVALID_DATA_CHANGE = 1002,
D3D12_MESSAGE_ID_DATA_STATIC_WHILE_SET_AT_EXECUTE_DESCRIPTOR_INVALID_DATA_CHANGE =
1003,
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_STATIC_DESCRIPTOR_DATA_STATIC_NOT_SET = 1004,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_ACCESS_OUT_OF_BOUNDS = 1005,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SAMPLER_MODE_MISMATCH = 1006,
D3D12_MESSAGE_ID_CREATE_FENCE_INVALID_FLAGS = 1007,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_DUPLICATE_SUBRESOURCE_TRANSITIONS = 1008,
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PRIORITY = 1009,
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_LARGE_NUM_DESCRIPTORS = 1013,
D3D12_MESSAGE_ID_BEGIN_EVENT = 1014,
D3D12_MESSAGE_ID_END_EVENT = 1015,
D3D12_MESSAGE_ID_CREATEDevice_DEBUG_LAYER_STARTUP_OPTIONS = 1016,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_DEPTHBOUNDSTEST_UNSUPPORTED = 1017,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_DUPLICATE_SUBOBJECT = 1018,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_UNKNOWN_SUBOBJECT = 1019,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_ZERO_SIZE_STREAM = 1020,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_STREAM = 1021,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CANNOT_DEDUCE_TYPE = 1022,
D3D12_MESSAGE_ID_COMMAND_LIST_STATIC_DESCRIPTOR_RESOURCE_DIMENSION_MISMATCH = 1023,
D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_PRIVILEGE_FOR_GLOBAL_REALTIME =
1024,

D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_HARDWARE_SUPPORT_FOR_GLOBAL_REALTI
ME = 1025,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_ARCHITECTURE = 1026,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DST = 1027,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE_DIMENSION = 1028,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DST_RANGE_OUT_OF_BOUNDS = 1029,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_SRC = 1030,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE_DIMENSION = 1031,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_SRC_RANGE_OUT_OF_BOUNDS = 1032,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_OFFSET_ALIGNMENT = 1033,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_RESOURCES = 1034,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_SUBRESOURCE_RANGES = 1035,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_RESOURCE = 1036,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_SUBRESOURCE_RANGE = 1037,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_SUBRESOURCE_OUT_OF_BOUNDS = 1038,
```

```
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_RANGE_OUT_OF_BOUNDS = 1039,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_ZERO_DEPENDENCIES = 1040,
D3D12_MESSAGE_ID_DEVICE_CREATE_SHARED_HANDLE_INVALIDARG = 1041,
D3D12_MESSAGE_ID_DESCRIPTOR_HANDLE_WITH_INVALID_RESOURCE = 1042,
D3D12_MESSAGE_ID_SETDEPTHBOUNDS_INVALIDARGS = 1043,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_STATE_IMPRECISE = 1044,
D3D12_MESSAGE_ID_COMMAND_LIST_PIPELINE_STATE_NOT_SET = 1045,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Model_MISMATCH = 1046,
D3D12_MESSAGE_ID_OBJECT_ACCESSED_WHILE_STILL_IN_USE = 1047,
D3D12_MESSAGE_ID_PROGRAMMABLE_MSAA_UNSUPPORTED = 1048,
D3D12_MESSAGE_ID_SETSAMPLEPOSITIONS_INVALIDARGS = 1049,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCEREGION_INVALID_RECT = 1050,
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDQUEUE = 1051,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDLIST = 1052,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDQUEUE = 1053,
D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDQUEUE = 1054,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDLIST = 1055,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDQUEUE = 1056,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDQUEUE = 1057,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDLIST = 1058,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDQUEUE = 1059,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSOR = 1060,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSSTREAM = 1061,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSOR = 1062,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSSTREAM = 1063,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSOR = 1064,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSSTREAM = 1065,
D3D12_MESSAGE_ID_PROCESS_FRAME_INVALID_PARAMETERS = 1066,
D3D12_MESSAGE_ID_COPY_INVALIDLAYOUT = 1067,
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION = 1068,
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION_POLICY = 1069,
D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION = 1070,
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION = 1071,
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION_POLICY = 1072,
D3D12_MESSAGE_ID_LIVE_PROTECTED_RESOURCE_SESSION = 1073,
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION = 1074,
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION_POLICY = 1075,
D3D12_MESSAGE_ID_DESTROY_PROTECTED_RESOURCE_SESSION = 1076,
D3D12_MESSAGE_ID_PROTECTED_RESOURCE_SESSION_UNSUPPORTED = 1077,
D3D12_MESSAGE_ID_FENCE_INVALIDOPERATION = 1078,
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_COPY_QUEUE_TIMESTAMPS_NOT_SUPPORTED = 1079,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_DEFERRED = 1080,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMFIRSTUSE = 1081,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMCLEAR = 1082,
D3D12_MESSAGE_ID_CREATE_VIDEODECODERHEAP = 1083,
D3D12_MESSAGE_ID_LIVE_VIDEODECODERHEAP = 1084,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODERHEAP = 1085,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDARG_RETURN = 1086,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_OUTOFMEMORY_RETURN = 1087,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDADDRESS = 1088,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDHANDLE = 1089,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_DEST = 1090,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_MODE = 1091,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_ALIGNMENT = 1092,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_NOT_SUPPORTED = 1093,
D3D12_MESSAGE_ID_SETVIEWINSTANCEMASK_INVALIDARGS = 1094,
D3D12_MESSAGE_ID_VIEW_INSTANCING_UNSUPPORTED = 1095,
D3D12_MESSAGE_ID_VIEW_INSTANCING_INVALIDARGS = 1096,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_DECODE_REFERENCE_ONLY_FLAG = 1097,
D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_DECODE_REFERENCE_ONLY_FLAG = 1098,
```

```
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_FAILURE = 1099,
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_UNSUPPORTED = 1100,
D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_INVALID_INPUT = 1101,
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODER_UNSUPPORTED = 1102,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Metadata_Error = 1103,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_View_Instancing_Vertex_Size_Exceeded =
1104,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Runtime_Internal_Error = 1105,
D3D12_MESSAGE_ID_No_Video_API_Support = 1106,
D3D12_MESSAGE_ID_Video_Process_Support_Invalid_Input = 1107,
D3D12_MESSAGE_ID_CREATE_Video_Processor_Caps_Failure = 1108,
D3D12_MESSAGE_ID_Video_Process_Support_Unsupported_Format = 1109,
D3D12_MESSAGE_ID_Video_Decode_Frame_Invalid_Argument = 1110,
D3D12_MESSAGE_ID_Enqueue_Make_Resident_Invalid_Flags = 1111,
D3D12_MESSAGE_ID_OpenExistingHeap_Unsupported = 1112,
D3D12_MESSAGE_ID_Video_Process_Frames_Invalid_Argument = 1113,
D3D12_MESSAGE_ID_Video_Decode_Support_Unsupported = 1114,
D3D12_MESSAGE_ID_CREATE_CommandRecorder = 1115,
D3D12_MESSAGE_ID_Live_CommandRecorder = 1116,
D3D12_MESSAGE_ID_Destroy_CommandRecorder = 1117,
D3D12_MESSAGE_ID_CREATE_Command_recorder_Video_Not_Supported = 1118,
D3D12_MESSAGE_ID_CREATE_Command_recorder_Invalid_Support_Flags = 1119,
D3D12_MESSAGE_ID_CREATE_Command_recorder_Invalid_Flags = 1120,
D3D12_MESSAGE_ID_CREATE_Command_recorder_More_Recorders_Than_Logical_Processors =
1121,
D3D12_MESSAGE_ID_CREATE_CommandPool = 1122,
D3D12_MESSAGE_ID_Live_CommandPool = 1123,
D3D12_MESSAGE_ID_Destroy_CommandPool = 1124,
D3D12_MESSAGE_ID_CREATE_Command_Pool_Invalid_Flags = 1125,
D3D12_MESSAGE_ID_CREATE_Command_List_Video_Not_Supported = 1126,
D3D12_MESSAGE_ID_Command_recorder_Support_Flags_Mismatch = 1127,
D3D12_MESSAGE_ID_Command_recorder_Contention = 1128,
D3D12_MESSAGE_ID_Command_recorder_Usage_With_CreateCommandList_Command_List = 1129,
D3D12_MESSAGE_ID_Command_Allocator_Usage_With_CreateCommandList1_Command_List =
1130,
D3D12_MESSAGE_ID_Cannot_Execute_Empty_Command_List = 1131,
D3D12_MESSAGE_ID_Cannot_Reset_Command_Pool_With_Open_Command_Lists = 1132,
D3D12_MESSAGE_ID_Cannot_Use_Command_recorder_Without_Current_Target = 1133,
D3D12_MESSAGE_ID_Cannot_Change_Command_recorder_Target_While_recording = 1134,
D3D12_MESSAGE_ID_Command_Pool_Sync = 1135,
D3D12_MESSAGE_ID_Evict_Underflow = 1136,
D3D12_MESSAGE_ID_CREATE_Meta_Command = 1137,
D3D12_MESSAGE_ID_Live_Meta_Command = 1138,
D3D12_MESSAGE_ID_Destroy_Meta_Command = 1139,
D3D12_MESSAGE_ID_CopyBufferRegion_Invalid_Dst_Resource = 1140,
D3D12_MESSAGE_ID_CopyBufferRegion_Invalid_Src_Resource = 1141,
D3D12_MESSAGE_ID_AtomicCopyBuffer_Invalid_Dst_Resource = 1142,
D3D12_MESSAGE_ID_AtomicCopyBuffer_Invalid_Src_Resource = 1143,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Null_Buffer = 1144,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Null_Resource_Desc = 1145,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Unsupported = 1146,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Invalid_Buffer_Dimension = 1147,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Invalid_Buffer_Flags = 1148,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Invalid_Buffer_Offset = 1149,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Invalid_Resource_Dimension = 1150,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_Invalid_Resource_Flags = 1151,
D3D12_MESSAGE_ID_CREATEPlacedResourceOnBuffer_OutOfMemory_Return = 1152,
D3D12_MESSAGE_ID_Cannot_Create_Graphics_and_Video_Command_recorder = 1153,
D3D12_MESSAGE_ID_UpdateTileMappings_Possibly_Mismatching_Properties = 1154,
D3D12_MESSAGE_ID_CREATE_Command_List_Invalid_Command_List_Type = 1155,
```

```
D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INCOMPATIBLE_WITH_STRUCTURED_BUFFERS =  
1156,  
D3D12_MESSAGE_ID_COMPUTE_ONLY_DEVICE_OPERATION_UNSUPPORTED = 1157,  
D3D12_MESSAGE_ID_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INVALID = 1158,  
D3D12_MESSAGE_ID_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_INVALID =  
1159,  
D3D12_MESSAGE_ID_COPY_RAYTRACING_ACCELERATION_STRUCTURE_INVALID = 1160,  
D3D12_MESSAGE_ID_DISPATCH_RAYS_INVALID = 1161,  
D3D12_MESSAGE_ID_GET_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO_INVALID =  
1162,  
D3D12_MESSAGE_ID_CREATE_LIFETIMETRACKER = 1163,  
D3D12_MESSAGE_ID_LIVE_LIFETIMETRACKER = 1164,  
D3D12_MESSAGE_ID_DESTROY_LIFETIMETRACKER = 1165,  
D3D12_MESSAGE_ID_DESTROYOWNEDOBJECT_OBJECTNOTOWNED = 1166,  
D3D12_MESSAGE_ID_CREATE_TRACKEDWORKLOAD = 1167,  
D3D12_MESSAGE_ID_LIVE_TRACKEDWORKLOAD = 1168,  
D3D12_MESSAGE_ID_DESTROY_TRACKEDWORKLOAD = 1169,  
D3D12_MESSAGE_ID_RENDER_PASS_ERROR = 1170,  
D3D12_MESSAGE_ID_META_COMMAND_ID_INVALID = 1171,  
D3D12_MESSAGE_ID_META_COMMAND_UNSUPPORTED_PARAMS = 1172,  
D3D12_MESSAGE_ID_META_COMMAND_FAILED_ENUMERATION = 1173,  
D3D12_MESSAGE_ID_META_COMMAND_PARAMETER_SIZE_MISMATCH = 1174,  
D3D12_MESSAGE_ID_UNINITIALIZED_META_COMMAND = 1175,  
D3D12_MESSAGE_ID_META_COMMAND_INVALID_GPU_VIRTUAL_ADDRESS = 1176,  
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDLIST = 1177,  
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDLIST = 1178,  
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDLIST = 1179,  
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDQUEUE = 1180,  
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDQUEUE = 1181,  
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDQUEUE = 1182,  
D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONESTIMATOR = 1183,  
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONESTIMATOR = 1184,  
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONESTIMATOR = 1185,  
D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONVECTORHEAP = 1186,  
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONVECTORHEAP = 1187,  
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONVECTORHEAP = 1188,  
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOADS = 1189,  
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOAD_PAIRS = 1190,  
D3D12_MESSAGE_ID_OUT_OF_ORDER_TRACKED_WORKLOAD_PAIR = 1191,  
D3D12_MESSAGE_ID_CANNOT_ADD_TRACKED_WORKLOAD = 1192,  
D3D12_MESSAGE_ID_INCOMPLETE_TRACKED_WORKLOAD_PAIR = 1193,  
D3D12_MESSAGE_ID_CREATE_STATE_OBJECT_ERROR = 1194,  
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_ERROR = 1195,  
D3D12_MESSAGE_ID_GET_SHADER_STACK_SIZE_ERROR = 1196,  
D3D12_MESSAGE_ID_GET_PIPELINE_STACK_SIZE_ERROR = 1197,  
D3D12_MESSAGE_ID_SET_PIPELINE_STACK_SIZE_ERROR = 1198,  
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_SIZE_INVALID = 1199,  
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_INVALID = 1200,  
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_DRIVER_REPORTED_ISSUE = 1201,  
D3D12_MESSAGE_ID_RENDER_PASS_INVALID_RESOURCE_BARRIER = 1202,  
D3D12_MESSAGE_ID_RENDER_PASS_DISALLOWED_API_CALLED = 1203,  
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_NEST_RENDER_PASSES = 1204,  
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_END_WITHOUT_BEGIN = 1205,  
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_CLOSE_COMMAND_LIST = 1206,  
D3D12_MESSAGE_ID_RENDER_PASS_GPU_WORK_WHILE_SUSPENDED = 1207,  
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_SUSPEND_RESUME = 1208,  
D3D12_MESSAGE_ID_RENDER_PASS_NO_PRIOR_SUSPEND_WITHIN_EXECUTECOMMANDLISTS = 1209,  
D3D12_MESSAGE_ID_RENDER_PASS_NO_SUBSEQUENT_RESUME_WITHIN_EXECUTECOMMANDLISTS =  
1210,  
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_COMMAND_QUEUE_MISMATCH = 1211,
```

```
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_NOT_SUPPORTED = 1212,
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_NO_ACCESS = 1213,
D3D12_MESSAGE_ID_RENDER_PASS_UNSUPPORTED_RESOLVE = 1214,
D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INVALID_RESOURCE_PTR = 1215,
D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_SIGNAL = 1216,
D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_WAIT = 1217,
D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_ESTIMATOR_INVALID_ARGUMENT = 1218,
D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT = 1219,
D3D12_MESSAGE_ID_ESTIMATE_MOTION_INVALID_ARGUMENT = 1220,
D3D12_MESSAGE_ID_RESOLVE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT = 1221,
D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_HEAP_TYPE = 1222,
D3D12_MESSAGE_ID_SET_BACKGROUND_PROCESSING_MODE_INVALID_ARGUMENT = 1223,
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE_FOR_FEATURE_LEVEL =
1224,
D3D12_MESSAGE_ID_CREATE_VIDEOEXTENSIONCOMMAND = 1225,
D3D12_MESSAGE_ID_LIVE_VIDEOEXTENSIONCOMMAND = 1226,
D3D12_MESSAGE_ID_DESTROY_VIDEOEXTENSIONCOMMAND = 1227,
D3D12_MESSAGE_ID_INVALID_VIDEO_EXTENSION_COMMAND_ID = 1228,
D3D12_MESSAGE_ID_VIDEO_EXTENSION_COMMAND_INVALID_ARGUMENT = 1229,
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_NOT_UNIQUE_IN_DXIL_LIBRARY = 1230,
D3D12_MESSAGE_ID_VARIABLE_SHADING_RATE_NOT_ALLOWED_WITH_TIR = 1231,

D3D12_MESSAGE_ID_GEOMETRY_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RAT
E_NOT_SUPPORTED_ON_DEVICE = 1232,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_SHADING_RATE = 1233,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_SHADING_RATE_NOT_PERMITTED_BY_CAP = 1234,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_COMBINER = 1235,
D3D12_MESSAGE_ID_RSSETSHADINGRATEIMAGE_REQUIRES_TIER_2 = 1236,
D3D12_MESSAGE_ID_RSSETSHADINGRATE_REQUIRES_TIER_1 = 1237,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_FORMAT = 1238,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_ARRAY_SIZE = 1239,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_MIP_LEVEL = 1240,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_COUNT = 1241,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_QUALITY = 1242,
D3D12_MESSAGE_ID_NON_RETAIL_SHADER_MODEL_WONT_VALIDATE = 1243,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_AS_ROOT_SIGNATURE_MISMATCH = 1244,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_ROOT_SIGNATURE_MISMATCH = 1245,
D3D12_MESSAGE_ID_ADD_TO_STATE_OBJECT_ERROR = 1246,
D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION_INVALID_ARGUMENT = 1247,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_PSO_DESC_MISMATCH = 1248,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_MS_INCOMPLETE_TYPE = 1249,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_AS_NOT_MS_MISMATCH = 1250,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_NOT_PS_MISMATCH = 1251,
D3D12_MESSAGE_ID_NONZERO_SAMPLER_FEEDBACK_MIP_REGION_WITH_INCOMPATIBLE_FORMAT =
1252,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_INPUTLAYOUT_SHADER_MISMATCH = 1253,
D3D12_MESSAGE_ID_EMPTY_DISPATCH = 1254,
D3D12_MESSAGE_ID_RESOURCE_FORMAT_REQUIRES_SAMPLER_FEEDBACK_CAPABILITY = 1255,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_MIP_REGION = 1256,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_DIMENSION = 1257,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_COUNT = 1258,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_QUALITY = 1259,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_LAYOUT = 1260,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_REQUIRES_UNORDERED_ACCESS_FLAG = 1261,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_NULL_ARGUMENTS = 1262,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_UAV_REQUIRES_SAMPLER_FEEDBACK_CAPABILITY = 1263,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_REQUIRES_FEEDBACK_MAP_FORMAT = 1264,
D3D12_MESSAGE_ID_CREATEMESHSHADER_INVALIDSHADERBYTECODE = 1265,
D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTOFMEMORY = 1266,
D3D12_MESSAGE_ID_CREATEMESHSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE = 1267,
```

```
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_INVALID_FORMAT =  
1268,  
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_INVALID_MIP_LEVEL_COUNT =  
1269,  
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_ARRAY_SIZE_MISMATCH  
= 1270,  
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_MISMATCHING_TARGETED_RESOURCE = 1271,  
D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTPUTEXCEEDSMAXSIZE = 1272,  
D3D12_MESSAGE_ID_CREATEMESHSHADER_GROUPSHAREDEXCEEDSMAXSIZE = 1273,  
  
D3D12_MESSAGE_ID_VERTEX_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_  
NOT_SUPPORTED_ON_DEVICE = 1274,  
  
D3D12_MESSAGE_ID_MESH_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NO  
T_SUPPORTED_ON_DEVICE = 1275,  
D3D12_MESSAGE_ID_CREATEMESHSHADER_MISMATCHEDASMSPAYLOADSIZE = 1276,  
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_UNBOUNDED_STATIC_DESCRIPTORS = 1277,  
D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_INVALIDSHADERBYTECODE = 1278,  
D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_OUTOFMEMORY = 1279,  
D3D12_MESSAGE_ID_CREATE_SHADERCACHESESSION = 1280,  
D3D12_MESSAGE_ID_LIVE_SHADERCACHESESSION = 1281,  
D3D12_MESSAGE_ID_DESTROY_SHADERCACHESESSION = 1282,  
D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_INVALIDARGS = 1283,  
D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_DISABLED = 1284,  
D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_ALREADYOPEN = 1285,  
D3D12_MESSAGE_ID_SHADERCACHECONTROL_DEVELOPERMODE = 1286,  
D3D12_MESSAGE_ID_SHADERCACHECONTROL_INVALIDFLAGS = 1287,  
D3D12_MESSAGE_ID_SHADERCACHECONTROL_STATEALREADYSET = 1288,  
D3D12_MESSAGE_ID_SHADERCACHECONTROL_IGNOREDFLAG = 1289,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_ALREADYPRESENT = 1290,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_HASHCOLLISION = 1291,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_CACHEFULL = 1292,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_FINDVALUE_NOTFOUND = 1293,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_CORRUPT = 1294,  
D3D12_MESSAGE_ID_SHADERCACHESESSION_DISABLED = 1295,  
D3D12_MESSAGE_ID_OVERSIZED_DISPATCH = 1296,  
D3D12_MESSAGE_ID_CREATE_VIDEOENCODER = 1297,  
D3D12_MESSAGE_ID_LIVE_VIDEOENCODER = 1298,  
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODER = 1299,  
D3D12_MESSAGE_ID_CREATE_VIDEOENCODERHEAP = 1300,  
D3D12_MESSAGE_ID_LIVE_VIDEOENCODERHEAP = 1301,  
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODERHEAP = 1302,  
D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_ENCODE_REFERENCE_ONLY_FLAG = 1303,  
D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_ENCODE_REFERENCE_ONLY_FLAG = 1304,  
D3D12_MESSAGE_ID_ENCODE_FRAME_INVALID_PARAMETERS = 1305,  
D3D12_MESSAGE_ID_ENCODE_FRAME_UNSUPPORTED_PARAMETERS = 1306,  
D3D12_MESSAGE_ID_RESOLVE_ENCODER_OUTPUT_METADATA_INVALID_PARAMETERS = 1307,  
D3D12_MESSAGE_ID_RESOLVE_ENCODER_OUTPUT_METADATA_UNSUPPORTED_PARAMETERS = 1308,  
D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_INVALID_PARAMETERS = 1309,  
D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_UNSUPPORTED_PARAMETERS = 1310,  
D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_HEAP_INVALID_PARAMETERS = 1311,  
D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_HEAP_UNSUPPORTED_PARAMETERS = 1312,  
D3D12_MESSAGE_ID_CREATECOMMANDLIST_NULL_COMMANDALLOCATOR,  
D3D12_MESSAGE_ID_CLEAR_UNORDERED_ACCESS_VIEW_INVALID_DESCRIPTOR_HANDLE,  
D3D12_MESSAGE_ID_DESCRIPTOR_HEAP_NOT_SHADER_VISIBLE,  
D3D12_MESSAGE_ID_CREATEBLENDSTATE_BLENDOP_WARNING,  
D3D12_MESSAGE_ID_CREATEBLENDSTATE_BLENDOPALPHA_WARNING,  
D3D12_MESSAGE_ID_WRITE_COMBINE_PERFORMANCE_WARNING,  
D3D12_MESSAGE_ID_RESOLVE_QUERY_INVALID_QUERY_STATE,  
D3D12_MESSAGE_ID_SETPRIVATEDATA_NO_ACCESS,
```

D3D12_MESSAGE_ID_COMMAND_LIST_STATIC_DESCRIPTOR_SAMPLER_MODE_MISMATCH,
D3D12_MESSAGE_ID_GETCOPYABLEFOOTPRINTS_UNSUPPORTED_BUFFER_WIDTH,
D3D12_MESSAGE_ID_CREATEMESHSHADER_TOPOLOGY_MISMATCH,
D3D12_MESSAGE_ID_VRS_SUM_COMBINER_REQUIRES_CAPABILITY,
D3D12_MESSAGE_ID_SETTING_SHADING_RATE_FROM_MS_REQUIRES_CAPABILITY,
D3D12_MESSAGE_ID_SHADERCACHESESSION_SHADERCACHEDELETE_NOTSUPPORTED,
D3D12_MESSAGE_ID_SHADERCACHECONTROL_SHADERCACHECLEAR_NOTSUPPORTED,
D3D12_MESSAGE_ID_CREATERESOURCE_STATE_IGNORED,
D3D12_MESSAGE_ID_UNUSED_CROSS_EXECUTE_SPLIT_BARRIER,
D3D12_MESSAGE_ID_DEVICE_OPEN_SHARED_HANDLE_ACCESS_DENIED,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_VALUES,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_ACCESS,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_SYNC,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_LAYOUT,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_TYPE,
D3D12_MESSAGE_ID_OUT_OF_BOUNDS_BARRIER_SUBRESOURCE_RANGE,
D3D12_MESSAGE_ID_INCOMPATIBLE_BARRIER_RESOURCE_DIMENSION,
D3D12_MESSAGE_ID_SET_SCISSOR_RECTS_INVALID_RECT,
D3D12_MESSAGE_ID_SHADING_RATE_SOURCE_REQUIRES_DIMENSION_TEXTURE2D,
D3D12_MESSAGE_ID_BUFFER_BARRIER_SUBREGION_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_UNSUPPORTED_BARRIER_LAYOUT,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALID_PARAMETERS,
D3D12_MESSAGE_ID_ENHANCED_BARRIERS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_LEGACY_BARRIER_VALIDATION_FORCED_ON,
D3D12_MESSAGE_ID_EMPTY_ROOT_DESCRIPTOR_TABLE,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ELEMENT_OFFSET_UNALIGNED,
D3D12_MESSAGE_ID_ALPHA_BLEND_FACTOR_NOT_SUPPORTED,
D3D12_MESSAGE_ID_BARRIER_INTEROP_INVALID_LAYOUT,
D3D12_MESSAGE_ID_BARRIER_INTEROP_INVALID_STATE,
D3D12_MESSAGE_ID_GRAPHICS_PIPELINE_STATE_DESC_ZERO_SAMPLE_MASK,
D3D12_MESSAGE_ID_INDEPENDENT_STENCIL_REF_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INDEPENDENT_MASKS_UNSUPPORTED,
D3D12_MESSAGE_ID_TEXTURE_BARRIER_SUBRESOURCES_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_NON_OPTIMAL_BARRIER_ONLY_EXECUTE_COMMAND_LISTS,
D3D12_MESSAGE_ID_EXECUTE_INDIRECT_ZERO_COMMAND_COUNT,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INCOMPATIBLE_TEXTURE_LAYOUT,
D3D12_MESSAGE_ID_DYNAMIC_INDEX_BUFFER_STRIP_CUT_NOT_SUPPORTED,
D3D12_MESSAGE_ID_PRIMITIVE_TOPOLOGY_TRIANGLE_FANS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATE_SAMPLER_COMPARISON_FUNC_IGNORED,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDHEAPTYPE,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPTYPE,
D3D12_MESSAGE_ID_DYNAMIC_DEPTH_BIAS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_NON_WHOLE_DYNAMIC_DEPTH_BIAS,
D3D12_MESSAGE_ID_DYNAMIC_DEPTH_BIAS_FLAG_MISSING,
D3D12_MESSAGE_ID_DYNAMIC_DEPTH_BIAS_NO_PIPELINE,
D3D12_MESSAGE_ID_DYNAMIC_INDEX_BUFFER_STRIP_CUT_FLAG_MISSING,
D3D12_MESSAGE_ID_DYNAMIC_INDEX_BUFFER_STRIP_CUT_NO_PIPELINE,
D3D12_MESSAGE_ID_NORMONORMALIZED_COORDINATE_SAMPLING_NOT_SUPPORTED,
D3D12_MESSAGE_ID_INVALID_CAST_TARGET,
D3D12_MESSAGE_ID_RENDER_PASS_COMMANDLIST_INVALID_END_STATE,
D3D12_MESSAGE_ID_RENDER_PASS_COMMANDLIST_INVALID_START_STATE,
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_ACCESS,
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_LOCAL_PRESERVE_PARAMETERS,
D3D12_MESSAGE_ID_RENDER_PASS_LOCAL_PRESERVE_RENDER_PARAMETERS_ERROR,
D3D12_MESSAGE_ID_RENDER_PASS_LOCAL_DEPTH_STENCIL_ERROR,
D3D12_MESSAGE_ID_DRAW_POTENTIALLY_OUTSIDE_OF_VALID_RENDER_AREA,
D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALID_LINERASTERIZATIONMODE,
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDALIGNMENT_SMALLRESOURCE,
D3D12_MESSAGE_ID_GENERIC_DEVICE_OPERATION_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_RENDER_TARGET_WRONG_WRITE_MASK,

```
D3D12_MESSAGE_ID_PROBABLE_PIX_EVENT_LEAK,  
D3D12_MESSAGE_ID_PIX_EVENT_UNDERFLOW,  
D3D12_MESSAGE_ID_RECREATEAT_INVALID_TARGET,  
D3D12_MESSAGE_ID_RECREATEAT_INSUFFICIENT_SUPPORT,  
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_STRUCTURED_BUFFER_STRIDE_MISMATCH,  
D3D12_MESSAGE_ID_D3D12_MESSAGES_END  
};
```

Constants

 [Expand table](#)

<code>D3D12_MESSAGE_ID_UNKNOWN</code>
Value: 0
<code>D3D12_MESSAGE_ID_STRING_FROM_APPLICATION</code>
Value: 1
<code>D3D12_MESSAGE_ID_CORRUPTED_THIS</code>
Value: 2
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER1</code>
Value: 3
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER2</code>
Value: 4
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER3</code>
Value: 5
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER4</code>
Value: 6
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETERS</code>
Value: 7
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER6</code>
Value: 8
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER7</code>
Value: 9
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER8</code>
Value: 10
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER9</code>
Value: 11
<code>D3D12_MESSAGE_ID_CORRUPTED_PARAMETER10</code>
Value: 12

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER11

Value: 13

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER12

Value: 14

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER13

Value: 15

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER14

Value: 16

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER15

Value: 17

D3D12_MESSAGE_ID_CORRUPTED_MULTITHREADING

Value: 18

D3D12_MESSAGE_ID_MESSAGE_REPORTING_OUTOFMEMORY

Value: 19

D3D12_MESSAGE_ID_GETPRIVATEDATA_MOREDATA

Value: 20

D3D12_MESSAGE_ID_SETPRIVATEDATA_INVALIDFREEDATA

Value: 21

D3D12_MESSAGE_ID_SETPRIVATEDATA_CHANGINGPARAMS

Value: 24

D3D12_MESSAGE_ID_SETPRIVATEDATA_OUTOFMEMORY

Value: 25

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_UNRECOGNIZEDFORMAT

Value: 26

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDESC

Value: 27

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFORMAT

Value: 28

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDVIDEOPLANESLICE

Value: 29

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDPLANESLICE

Value: 30

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDIMENSIONS

Value: 31

D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDRESOURCE

Value: 32

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNRECOGNIZEDFORMAT

Value: 35

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNSUPPORTEDFORMAT

Value: 36

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDESC

Value: 37

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDFORMAT

Value: 38

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDVIDEOPLANESLICE

Value: 39

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDPLANESLICE

Value: 40

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDIMENSIONS

Value: 41

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDRESOURCE

Value: 42

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_UNRECOGNIZEDFORMAT

Value: 45

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDESC

Value: 46

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFORMAT

Value: 47

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDIMENSIONS

Value: 48

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDRESOURCE

Value: 49

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_OUTOFMEMORY

Value: 52

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TOOMANYELEMENTS

Value: 53

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDFORMAT

Value: 54

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INCOMPATIBLEFORMAT

Value: 55

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOT

Value: 56

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDINPUTSLOTCLASS

Value: 57

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_STEPSLOTCLASSMISMATCH

Value: 58

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSLOTCLASSCHANGE

Value: 59

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSTEPRATECHANGE

Value: 60

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDALIGNMENT

Value: 61

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_DUPLICATESEMANTIC

Value: 62

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_UNPARSEABLEINPUTSIGNATURE

Value: 63

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_NULLSEMANTIC

Value: 64

D3D12_MESSAGE_ID_CREATEVERTEXSHADER_OUTOFMEMORY

Value: 65

D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERBYTECODE

Value: 67

D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERTYPE

Value: 68

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_OUTOFMEMORY

Value: 69

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERBYTECODE

Value: 70

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERTYPE

Value: 71

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTOFMEMORY

Value: 72

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERBYTECODE

Value: 73

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE

Value: 74

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMENTRIES

Value: 75

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSTREAMSTRIDEUNUSED

Value: 76

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSLOT0EXPECTED

Value: 79

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSLOT

Value: 80

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_ONLYONEELEMENTPERSLOT

Value: 81

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCOMPONENTCOUNT

Value: 82

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTARTCOMPONENTANDCOMPONENTCOUNT

Value: 83

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDGAPDEFINITION

Value: 84

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_REPEATEDOUTPUT

Value: 85

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSTREAMSTRIDE

Value: 86

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGSEMANTIC

Value: 87

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MASKMISMATCH

Value: 88

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_CANTHAVEONLYGAPS

Value: 89

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DECLTOOCOMPLEX

Value: 90

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGOUTPUTSIGNATURE

Value: 91

D3D12_MESSAGE_ID_CREATEPIXELSHADER_OUTOFMEMORY

Value: 92

D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERBYTECODE

Value: 93

D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERTYPE

Value: 94

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDFILLMODE

Value: 95

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDCULLMODE

Value: 96

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDDEPTHBIASCLAMP

Value: 97

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDSLOPESCALEDDEPTHBIAS

Value: 98

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHWRITEMASK

Value: 100

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHFUNC

Value: 101

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTPAUSESTENCILFAILOP

Value: 102

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTPAUSESTENCILZFAILOP

Value: 103

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTPAUSESTENCILPASSOP

Value: 104

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTPAUSESTENCILFUNC

Value: 105

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKPAUSESTENCILFAILOP

Value: 106

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKPAUSESTENCILZFAILOP

Value: 107

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKPAUSESTENCILPASSOP

Value: 108

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKPAUSESTENCILFUNC

Value: 109

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND

Value: 111

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEND

Value: 112

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP

Value: 113

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLENDALPHA

Value: 114

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLENDALPHA
Value: 115
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOPALPHA
Value: 116
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDRENDERTARGETWRITEMASK
Value: 117
D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_INVALID
Value: 135
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_NOT_SET
Value: 200
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_MISMATCH
Value: 201
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_NOT_SET
Value: 202
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_STRIDE_TOO_SMALL
Value: 209
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_TOO_SMALL
Value: 210
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_NOT_SET
Value: 211
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_FORMAT_INVALID
Value: 212
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_TOO_SMALL
Value: 213
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INVALID_PRIMITIVE_TOPOLOGY
Value: 219
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_STRIDE_UNALIGNED
Value: 221
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_OFFSET_UNALIGNED
Value: 222
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_AT_FAULT
Value: 232
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_POSSIBLY_AT_FAULT
Value: 233
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_NOT_AT_FAULT
Value: 234

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TRAILING_DIGIT_IN_SEMANTIC

Value: 239

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_TRAILING_DIGIT_IN_SEMANTIC

Value: 240

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TYPE_MISMATCH

Value: 245

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_EMPTY_LAYOUT

Value: 253

D3D12_MESSAGE_ID_LIVE_OBJECT_SUMMARY

Value: 255

D3D12_MESSAGE_ID_LIVE_DEVICE

Value: 274

D3D12_MESSAGE_ID_LIVE_SWAPCHAIN

Value: 275

D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFLAGS

Value: 276

D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDCLASSLINKAGE

Value: 277

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDCLASSLINKAGE

Value: 278

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAMTORASTERIZER

Value: 280

D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDCLASSLINKAGE

Value: 283

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAM

Value: 284

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDENTRIES

Value: 285

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDSTRIDES

Value: 286

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMSTRIDES

Value: 287

D3D12_MESSAGE_ID_CREATEHULLSHADER_OUTOFMEMORY

Value: 289

D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERBYTECODE

Value: 290

D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERTYPE

Value: 291

D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDCLASSLINKAGE

Value: 292

D3D12_MESSAGE_ID_CREATEDOMAINSHADER_OUTOFMEMORY

Value: 294

D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERBYTECODE

Value: 295

D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERTYPE

Value: 296

D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCLASSLINKAGE

Value: 297

D3D12_MESSAGE_ID_RESOURCE_UNMAP_NOTMAPPED

Value: 310

D3D12_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_MISMATCHED_DATA_SIZE

Value: 318

D3D12_MESSAGE_ID_CREATECOMPUTESHADER_OUTOFMEMORY

Value: 321

D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERBYTECODE

Value: 322

D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCLASSLINKAGE

Value: 323

D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 331

D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 332

D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 333

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 334

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEFLOATOPSNOTSUPPORTED

Value: 335

D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 336

D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEFLOATOPSNOTSUPPORTED

Value: 337

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDRESOURCE

Value: 340

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDESC

Value: 341

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFORMAT

Value: 342

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDVIDEOPLANESLICE

Value: 343

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDPLANESLICE

Value: 344

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDIMENSIONS

Value: 345

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_UNRECOGNIZEDFORMAT

Value: 346

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFLAGS

Value: 354

D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFORCEDSAMPLECOUNT

Value: 401

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDLOGICOPS

Value: 403

D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 410

D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 412

D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 414

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 416

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 418

D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 420

D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEEXTENSIONSNOTSUPPORTED

Value: 422

D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_UAVSNOTSUPPORTED

Value: 425

D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_UAVSNOTSUPPORTED

Value: 426

D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_UAVSNOTSUPPORTED

Value: 427

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_UAVSNOTSUPPORTED

Value: 428

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UAVSNOTSUPPORTED

Value: 429

D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_UAVSNOTSUPPORTED

Value: 430

D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_UAVSNOTSUPPORTED

Value: 431

D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_INVALIDSOURCERECT

Value: 447

D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_EMPTYRECT

Value: 448

D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_INVALID_PARAMETER

Value: 493

D3D12_MESSAGE_ID_COPYTILEMAPPINGS_INVALID_PARAMETER

Value: 494

D3D12_MESSAGE_ID_CREATEDevice_INVALIDARGS

Value: 506

D3D12_MESSAGE_ID_CREATEDevice_WARNING

Value: 507

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_TYPE

Value: 519

D3D12_MESSAGE_ID_RESOURCE_BARRIER_NULL_POINTER

Value: 520

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SUBRESOURCE

Value: 521

D3D12_MESSAGE_ID_RESOURCE_BARRIER_RESERVED_BITS

Value: 522

D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISSING_BIND_FLAGS

Value: 523

D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_MISC_FLAGS

Value: 524

D3D12_MESSAGE_ID_RESOURCE_BARRIER_MATCHING_STATES

Value: 525

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINATION

Value: 526

D3D12_MESSAGE_ID_RESOURCE_BARRIER_BEFORE_AFTER_MISMATCH

Value: 527

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_RESOURCE

Value: 528

D3D12_MESSAGE_ID_RESOURCE_BARRIER_SAMPLE_COUNT

Value: 529

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS

Value: 530

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINED_FLAGS

Value: 531

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS_FOR_FORMAT

Value: 532

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SPLIT_BARRIER

Value: 533

D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_END

Value: 534

D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_BEGIN

Value: 535

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAG

Value: 536

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMMAND_LIST_TYPE

Value: 537

D3D12_MESSAGE_ID_INVALID_SUBRESOURCE_STATE

Value: 538

D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CONTENTION

Value: 540

D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET

Value: 541

D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET_BUNDLE

Value: 542

D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CANNOT_RESET

Value: 543

D3D12_MESSAGE_ID_COMMAND_LIST_OPEN

Value: 544

D3D12_MESSAGE_ID_INVALID_BUNDLE_API

Value: 546

D3D12_MESSAGE_ID_COMMAND_LIST_CLOSED

Value: 547

D3D12_MESSAGE_ID_WRONG_COMMAND_ALLOCATOR_TYPE

Value: 549

D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_SYNC

Value: 552

D3D12_MESSAGE_ID_COMMAND_LIST_SYNC

Value: 553

D3D12_MESSAGE_ID_SET_DESCRIPTOR_HEAP_INVALID

Value: 554

D3D12_MESSAGE_ID_CREATE_COMMANDQUEUE

Value: 557

D3D12_MESSAGE_ID_CREATE_COMMANDALLOCATOR

Value: 558

D3D12_MESSAGE_ID_CREATE_PIPELINESTATE

Value: 559

D3D12_MESSAGE_ID_CREATE_COMMANDLIST12

Value: 560

D3D12_MESSAGE_ID_CREATE_RESOURCE

Value: 562

D3D12_MESSAGE_ID_CREATE_DESCRIPTORHEAP

Value: 563

D3D12_MESSAGE_ID_CREATE_ROOTSIGNATURE

Value: 564

D3D12_MESSAGE_ID_CREATE_LIBRARY

Value: 565

D3D12_MESSAGE_ID_CREATE_HEAP

Value: 566

D3D12_MESSAGE_ID_CREATE_MONITOREDFENCE

Value: 567

D3D12_MESSAGE_ID_CREATE_QUERYHEAP

Value: 568

D3D12_MESSAGE_ID_CREATE_COMMANDSIGNATURE

Value: 569

D3D12_MESSAGE_ID_LIVE_COMMANDQUEUE

Value: 570

D3D12_MESSAGE_ID_LIVE_COMMANDALLOCATOR

Value: 571

D3D12_MESSAGE_ID_LIVE_PIPELINESTATE

Value: 572

D3D12_MESSAGE_ID_LIVE_COMMANDLIST12

Value: 573

D3D12_MESSAGE_ID_LIVE_RESOURCE

Value: 575

D3D12_MESSAGE_ID_LIVE_DESCRIPTORHEAP

Value: 576

D3D12_MESSAGE_ID_LIVE_ROOTSIGNATURE

Value: 577

D3D12_MESSAGE_ID_LIVE_LIBRARY

Value: 578

D3D12_MESSAGE_ID_LIVE_HEAP

Value: 579

D3D12_MESSAGE_ID_LIVE_MONITOREDFENCE

Value: 580

D3D12_MESSAGE_ID_LIVE_QUERYHEAP

Value: 581

D3D12_MESSAGE_ID_LIVE_COMMANDSIGNATURE

Value: 582

D3D12_MESSAGE_ID_DESTROY_COMMANDQUEUE

Value: 583

D3D12_MESSAGE_ID_DESTROY_COMMANDALLOCATOR

Value: 584

D3D12_MESSAGE_ID_DESTROY_PIPELINESTATE

Value: 585

D3D12_MESSAGE_ID_DESTROY_COMMANDLIST12

Value: 586

D3D12_MESSAGE_ID_DESTROY_RESOURCE

Value: 588

D3D12_MESSAGE_ID_DESTROY_DESCRIPTORHEAP

Value: 589

D3D12_MESSAGE_ID_DESTROY_ROOTSIGNATURE

Value: 590

D3D12_MESSAGE_ID_DESTROY_LIBRARY

Value: 591

D3D12_MESSAGE_ID_DESTROY_HEAP

Value: 592

D3D12_MESSAGE_ID_DESTROY_MONITOREDFENCE

Value: 593

D3D12_MESSAGE_ID_DESTROY_QUERYHEAP

Value: 594

D3D12_MESSAGE_ID_DESTROY_COMMANDSIGNATURE

Value: 595

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDIMENSIONS

Value: 597

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDMISCFLAGS

Value: 599

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDARG_RETURN

Value: 602

D3D12_MESSAGE_ID_CREATERESOURCE_OUTOFMEMORY_RETURN

Value: 603

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDESC

Value: 604

D3D12_MESSAGE_ID_POSSIBLY_INVALID_SUBRESOURCE_STATE

Value: 607

D3D12_MESSAGE_ID_INVALID_USE_OF_NON_RESIDENT_RESOURCE

Value: 608

D3D12_MESSAGE_ID_POSSIBLE_INVALID_USE_OF_NON_RESIDENT_RESOURCE

Value: 609

D3D12_MESSAGE_ID_BUNDLE_PIPELINE_STATE_MISMATCH

Value: 610

D3D12_MESSAGE_ID_PRIMITIVE_TOPOLOGY_MISMATCH_PIPELINE_STATE

Value: 611

D3D12_MESSAGE_ID_RENDER_TARGET_FORMAT_MISMATCH_PIPELINE_STATE

Value: 613

D3D12_MESSAGE_ID_RENDER_TARGET_SAMPLE_DESC_MISMATCH_PIPELINE_STATE

Value: 614

D3D12_MESSAGE_ID_DEPTH_STENCIL_FORMAT_MISMATCH_PIPELINE_STATE

Value: 615

D3D12_MESSAGE_ID_DEPTH_STENCIL_SAMPLE_DESC_MISMATCH_PIPELINE_STATE

Value: 616

D3D12_MESSAGE_ID_CREATESHADER_INVALIDBYTECODE

Value: 622

D3D12_MESSAGE_ID_CREATEHEAP_NULLDESC

Value: 623

D3D12_MESSAGE_ID_CREATEHEAP_INVALIDSIZE

Value: 624

D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDHEAPTYPE

Value: 625

D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES

Value: 626

D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMEMORYPOOL

Value: 627

D3D12_MESSAGE_ID_CREATEHEAP_INVALIDPROPERTIES

Value: 628

D3D12_MESSAGE_ID_CREATEHEAP_INVALIDALIGNMENT

Value: 629

D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMISCFLAGS

Value: 630

D3D12_MESSAGE_ID_CREATEHEAP_INVALIDMISCFLAGS

Value: 631

D3D12_MESSAGE_ID_CREATEHEAP_INVALIDARG_RETURN

Value: 632

D3D12_MESSAGE_ID_CREATEHEAP_OUTOFMEMORY_RETURN

Value: 633

D3D12_MESSAGE_ID_CREATEROBJECTANDHEAP_NULLHEAPPROPERTIES

Value: 634

D3D12_MESSAGE_ID_CREATEROBJECTANDHEAP_UNRECOGNIZEDHEAPTYPE

Value: 635

D3D12_MESSAGE_ID_CREATEROBJECTANDHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES

Value: 636

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDMEMORYPOOL

Value: 637

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPPROPERTIES

Value: 638

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPMISCFLAGS

Value: 639

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPMISCFLAGS

Value: 640

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDARG_RETURN

Value: 641

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_OUTOFGMEMORY_RETURN

Value: 642

D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_UNRECOGNIZEDHEAPTYPE

Value: 643

D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_INVALIDHEAPTYPE

Value: 644

D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_INVALID_DESC

Value: 645

D3D12_MESSAGE_ID_INVALID_DESCRIPTOR_HANDLE

Value: 646

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALID_CONSERVATIVERASTERMODE

Value: 647

D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_RESOURCE

Value: 649

D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_DESC

Value: 650

D3D12_MESSAGE_ID_CREATE_UNORDEREDACCESS_VIEW_INVALID_COUNTER_USAGE

Value: 652

D3D12_MESSAGE_ID_COPY_DESCRIPTORS_INVALID_RANGES

Value: 653

D3D12_MESSAGE_ID_COPY_DESCRIPTORS_WRITE_ONLY_DESCRIPTOR

Value: 654

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_RTV_FORMAT_NOT_UNKNOWN

Value: 655

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_RENDER_TARGET_COUNT

Value: 656

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_VERTEX_SHADER_NOT_SET

Value: 657

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INPUTLAYOUT_NOT_SET

Value: 658

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_HS_DS_SIGNATURE_MISMATCH

Value: 659

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_REGISTERINDEX

Value: 660

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_COMPONENTTYPE

Value: 661

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_REGISTERMASK

Value: 662

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_SYSTEMVALUE

Value: 663

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_NEVERWRITTEN_ALWAYSREADS

Value: 664

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_MINPRECISION

Value: 665

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_SHADER_LINKAGE_SEMANTICNAME_NOT_FOUND

Value: 666

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HS_XOR_DS_MISMATCH

Value: 667

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HULL_SHADER_INPUT_TOPOLOGY_MISMATCH

Value: 668

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HS_DS_CONTROL_POINT_COUNT_MISMATCH

Value: 669

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HS_DS_TESSELLATOR_DOMAIN_MISMATCH

Value: 670

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_USE_OF_CENTER_MULTISAMPLE_PATTERN

Value: 671

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_USE_OF_FORCED_SAMPLE_COUNT

Value: 672

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_PRIMITIVE_TOPOLOGY

Value: 673

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_SYSTEMVALUE

Value: 674

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_OM_DUAL_SOURCE_BLENDING_CAN_ONLY_HAVE_RENDER_TARGET_0
Value: 675
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_OM_RENDER_TARGET_DOES_NOT_SUPPORT_BLENDING
Value: 676
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_PS_OUTPUT_TYPE_MISMATCH
Value: 677
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_OM_RENDER_TARGET_DOES_NOT_SUPPORT_LOGIC_OPS
Value: 678
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_RENDERTARGETVIEW_NOT_SET
Value: 679
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_DEPTHSTENCILVIEW_NOT_SET
Value: 680
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_GS_INPUT_PRIMITIVE_MISMATCH
Value: 681
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_POSITION_NOT_PRESENT
Value: 682
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_MISSING_ROOT_SIGNATURE_FLAGS
Value: 683
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_INDEX_BUFFER_PROPERTIES
Value: 684
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVALID_SAMPLE_DESC
Value: 685
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HS_ROOT_SIGNATURE_MISMATCH
Value: 686
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_DS_ROOT_SIGNATURE_MISMATCH
Value: 687
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_VS_ROOT_SIGNATURE_MISMATCH
Value: 688
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_GS_ROOT_SIGNATURE_MISMATCH
Value: 689
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_PS_ROOT_SIGNATURE_MISMATCH
Value: 690
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_MISSING_ROOT_SIGNATURE
Value: 691
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_OPEN_BUNDLE
Value: 692

D3D12_MESSAGE_ID_EXECUTE_BUNDLE_DESCRIPTOR_HEAP_MISMATCH

Value: 693

D3D12_MESSAGE_ID_EXECUTE_BUNDLE_TYPE

Value: 694

D3D12_MESSAGE_ID_DRAW_EMPTY_SCISSOR_RECTANGLE

Value: 695

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_BLOB_NOT_FOUND

Value: 696

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_DESERIALIZE_FAILED

Value: 697

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_INVALID_CONFIGURATION

Value: 698

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_NOT_SUPPORTED_ON_DEVICE

Value: 699

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLRESOURCEPROPERTIES

Value: 700

D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLHEAP

Value: 701

D3D12_MESSAGE_ID_GETRESOURCEALLOCATIONINFO_INVALIDRDESCS

Value: 702

D3D12_MESSAGE_ID_MAKERESIDENT_NULLOBJECTARRAY

Value: 703

D3D12_MESSAGE_ID_EVICT_NULLOBJECTARRAY

Value: 705

D3D12_MESSAGE_ID_SET_DESCRIPTOR_TABLE_INVALID

Value: 708

D3D12_MESSAGE_ID_SET_ROOT_CONSTANT_INVALID

Value: 709

D3D12_MESSAGE_ID_SET_ROOT_CONSTANT_BUFFER_VIEW_INVALID

Value: 710

D3D12_MESSAGE_ID_SET_ROOT_SHADER_RESOURCE_VIEW_INVALID

Value: 711

D3D12_MESSAGE_ID_SET_ROOT_UNORDERED_ACCESS_VIEW_INVALID

Value: 712

D3D12_MESSAGE_ID_SET_VERTEX_BUFFERS_INVALID_DESC

Value: 713

D3D12_MESSAGE_ID_SET_INDEX_BUFFER_INVALID_DESC

Value: 715

D3D12_MESSAGE_ID_SET_STREAM_OUTPUT_BUFFERS_INVALID_DESC

Value: 717

D3D12_MESSAGE_ID_CREATERO RESOURCE_UNRECOGNIZEDDIMENSIONALITY

Value: 718

D3D12_MESSAGE_ID_CREATERO RESOURCE_UNRECOGNIZEDLAYOUT

Value: 719

D3D12_MESSAGE_ID_CREATERO RESOURCE_INVALIDDIMENSIONALITY

Value: 720

D3D12_MESSAGE_ID_CREATERO RESOURCE_INVALIDALIGNMENT

Value: 721

D3D12_MESSAGE_ID_CREATERO RESOURCE_INVALIDMIPLEVELS

Value: 722

D3D12_MESSAGE_ID_CREATERO RESOURCE_INVALIDSAMPLEDESC

Value: 723

D3D12_MESSAGE_ID_CREATERO RESOURCE_INVALIDLAYOUT

Value: 724

D3D12_MESSAGE_ID_SET_INDEX_BUFFER_INVALID

Value: 725

D3D12_MESSAGE_ID_SET_VERTEX_BUFFERS_INVALID

Value: 726

D3D12_MESSAGE_ID_SET_STREAM_OUTPUT_BUFFERS_INVALID

Value: 727

D3D12_MESSAGE_ID_SET_RENDER_TARGETS_INVALID

Value: 728

D3D12_MESSAGE_ID_CREATEQUERY_HEAP_INVALID_PARAMETERS

Value: 729

D3D12_MESSAGE_ID_BEGIN_END_QUERY_INVALID_PARAMETERS

Value: 731

D3D12_MESSAGE_ID_CLOSE_COMMAND_LIST_OPEN_QUERY

Value: 732

D3D12_MESSAGE_ID_RESOLVE_QUERY_DATA_INVALID_PARAMETERS

Value: 733

D3D12_MESSAGE_ID_SET_PREDICATION_INVALID_PARAMETERS

Value: 734

D3D12_MESSAGE_ID_TIMESTAMPS_NOT_SUPPORTED

Value: 735

D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDFORMAT

Value: 737

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDFORMAT

Value: 738

D3D12_MESSAGE_ID_GETCOPYABLEFOOTPRINTS_INVALIDSUBRESOURCERANGE

Value: 739

D3D12_MESSAGE_ID_GETCOPYABLEFOOTPRINTS_INVALIDBASEOFFSET

Value: 740

D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDSUBRESOURCERANGE

Value: 739

D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDBASEOFFSET

Value: 740

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_HEAP

Value: 741

D3D12_MESSAGE_ID_CREATE_SAMPLER_INVALID

Value: 742

D3D12_MESSAGE_ID_CREATECOMMANDSIGNATURE_INVALID

Value: 743

D3D12_MESSAGE_ID_EXECUTE_INDIRECT_INVALID_PARAMETERS

Value: 744

D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_RESOURCE_DIMENSION

Value: 745

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDCLEARVALUE

Value: 815

D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDCLEARVALUEFORMAT

Value: 816

D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDCLEARVALUEFORMAT

Value: 817

D3D12_MESSAGE_ID_CREATERESOURCE_CLEARVALUEDENORMFLUSH

Value: 818

D3D12_MESSAGE_ID_CLEARRENDERTARGETVIEW_MISMATCHINGCLEARVALUE

Value: 820

D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_MISMATCHINGCLEARVALUE

Value: 821

D3D12_MESSAGE_ID_MAP_INVALIDHEAP

Value: 822

D3D12_MESSAGE_ID_UNMAP_INVALIDHEAP

Value: 823

D3D12_MESSAGE_ID_MAP_INVALIDRESOURCE

Value: 824

D3D12_MESSAGE_ID_UNMAP_INVALIDRESOURCE

Value: 825

D3D12_MESSAGE_ID_MAP_INVALIDSUBRESOURCE

Value: 826

D3D12_MESSAGE_ID_UNMAP_INVALIDSUBRESOURCE

Value: 827

D3D12_MESSAGE_ID_MAP_INVALIDDRANGE

Value: 828

D3D12_MESSAGE_ID_UNMAP_INVALIDDRANGE

Value: 829

D3D12_MESSAGE_ID_MAP_INVALIDDATAPORTER

Value: 832

D3D12_MESSAGE_ID_MAP_INVALIDARG_RETURN

Value: 833

D3D12_MESSAGE_ID_MAP_OUTOFMEMORY_RETURN

Value: 834

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_BUNDLENOTSUPPORTED

Value: 835

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_COMMANDLISTMISMATCH

Value: 836

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_OPENCOMMANDLIST

Value: 837

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_FAILEDCOMMANDLIST

Value: 838

D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLDST

Value: 839

D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDDSTRESOURCEDIMENSION

Value: 840

D3D12_MESSAGE_ID_COPYBUFFERREGION_DSTRANGEOUTOFCOMMANDLIST

Value: 841

D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLSRC
Value: 842
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDSRCRESOURCEDIMENSION
Value: 843
D3D12_MESSAGE_ID_COPYBUFFERREGION_SRCRANGEOUTOFCOMMANDLIST
Value: 844
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDCOPYFLAGS
Value: 845
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLDST
Value: 846
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDSTTYPE
Value: 847
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRESOURCEDIMENSION
Value: 848
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRESOURCE
Value: 849
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTSUBRESOURCE
Value: 850
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTOFFSET
Value: 851
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDSTFORMAT
Value: 852
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTFORMAT
Value: 853
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDIMENSIONS
Value: 854
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTROWPITCH
Value: 855
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTPLACEMENT
Value: 856
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDPLACEDFOOTPRINTFORMAT
Value: 857
D3D12_MESSAGE_ID_COPYTEXTUREREGION_DSTREGIONOUTOFCOMMANDLIST
Value: 858
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLSRC
Value: 859

D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCTYPE

Value: 860

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCEDIMENSION

Value: 861

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCE

Value: 862

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCSUBRESOURCE

Value: 863

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCOFFSET

Value: 864

D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCFORMAT

Value: 865

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCFORMAT

Value: 866

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDIMENSIONS

Value: 867

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCROWPITCH

Value: 868

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCPLACEMENT

Value: 869

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDSPLACEDFOOTPRINTFORMAT

Value: 870

D3D12_MESSAGE_ID_COPYTEXTUREREGION_SRCREGIONOUTOFCOMMITS

Value: 871

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTCOORDINATES

Value: 872

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCBOX

Value: 873

D3D12_MESSAGE_ID_COPYTEXTUREREGION_FORMATMISMATCH

Value: 874

D3D12_MESSAGE_ID_COPYTEXTUREREGION_EMPTYBOX

Value: 875

D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDCOPYFLAGS

Value: 876

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SUBRESOURCE_INDEX

Value: 877

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_FORMAT

Value: 878

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_MISMATCH

Value: 879

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SAMPLE_COUNT

Value: 880

D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_INVALID_SHADER

Value: 881

D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_CS_ROOT_SIGNATURE_MISMATCH

Value: 882

D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_MISSING_ROOT_SIGNATURE

Value: 883

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALIDCACHEDBLOB

Value: 884

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBADAPTERMISMATCH

Value: 885

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDRIVERVERSIONMISMATCH

Value: 886

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDESCMISMATCH

Value: 887

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBIGNORED

Value: 888

D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDHEAP

Value: 889

D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDRESOURCE

Value: 890

D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDBOX

Value: 891

D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDSUBRESOURCE

Value: 892

D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_EMPTYBOX

Value: 893

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDHEAP

Value: 894

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDRESOURCE

Value: 895

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDBOX

Value: 896

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDSUBRESOURCE

Value: 897

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_EMPTYBOX

Value: 898

D3D12_MESSAGE_ID_TOO_MANY_NODES_SPECIFIED

Value: 899

D3D12_MESSAGE_ID_INVALID_NODE_INDEX

Value: 900

D3D12_MESSAGE_ID_GETTHEAPPROPRIETIES_INVALIDRESOURCE

Value: 901

D3D12_MESSAGE_ID_NODE_MASK_MISMATCH

Value: 902

D3D12_MESSAGE_ID_COMMAND_LIST_OUTOFMEMORY

Value: 903

D3D12_MESSAGE_ID_COMMAND_LIST_MULTIPLE_SWAPCHAIN_BUFFER_REFERENCES

Value: 904

D3D12_MESSAGE_ID_COMMAND_LIST_TOO_MANY_SWAPCHAIN_REFERENCES

Value: 905

D3D12_MESSAGE_ID_COMMAND_QUEUE_TOO_MANY_SWAPCHAIN_REFERENCES

Value: 906

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_WRONGSWAPCHAINBUFFERREFERENCE

Value: 907

D3D12_MESSAGE_ID_COMMAND_LIST_SETRENDERTARGETS_INVALIDNUMRENDERTARGETS

Value: 908

D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_TYPE

Value: 909

D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_FLAGS

Value: 910

D3D12_MESSAGE_ID_CREATESHAREDRESOURCE_INVALIDFLAGS

Value: 911

D3D12_MESSAGE_ID_CREATESHAREDRESOURCE_INVALIDFORMAT

Value: 912

D3D12_MESSAGE_ID_CREATESHAREDHEAP_INVALIDFLAGS

Value: 913

D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_UNRECOGNIZEDPROPERTIES

Value: 914

D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDSIZE

Value: 915

D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDOBJECT

Value: 916

D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDOBJECT

Value: 917

D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDKEY

Value: 918

D3D12_MESSAGE_ID_KEYEDMUTEX_WRONGSTATE

Value: 919

D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_PRIORITY

Value: 920

D3D12_MESSAGE_ID_OBJECT_DELETED_WHILE_STILL_IN_USE

Value: 921

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_FLAGS

Value: 922

D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_HAS_NO_RESOURCE

Value: 923

D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_RENDER_TARGET_DELETED

Value: 924

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_ALL_RENDER_TARGETS_HAVE_UNKNOWN_FORMAT

Value: 925

D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_INTERSECTS_MULTIPLE_BUFFERS

Value: 926

D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_GPU_WRITTEN_READBACK_RESOURCE_MAPPED

Value: 927

D3D12_MESSAGE_ID_UNMAP_RANGE_NOT_EMPTY

Value: 929

D3D12_MESSAGE_ID_MAP_INVALID_NULLRANGE

Value: 930

D3D12_MESSAGE_ID_UNMAP_INVALID_NULLRANGE

Value: 931

D3D12_MESSAGE_ID_NO_GRAPHICS_API_SUPPORT

Value: 932

D3D12_MESSAGE_ID_NO_COMPUTE_API_SUPPORT

Value: 933

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_FLAGS_NOT_SUPPORTED

Value: 934

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_ARGUMENT_UNINITIALIZED

Value: 935

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_HEAP_INDEX_OUT_OF_BOUNDS

Value: 936

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TABLE_REGISTER_INDEX_OUT_OF_BOUNDS

Value: 937

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_UNINITIALIZED

Value: 938

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TYPE_MISMATCH

Value: 939

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SRV_RESOURCE_DIMENSION_MISMATCH

Value: 940

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UAV_RESOURCE_DIMENSION_MISMATCH

Value: 941

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INCOMPATIBLE_RESOURCE_STATE

Value: 942

D3D12_MESSAGE_ID_COPYRESOURCE_NULLDST

Value: 943

D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDDSTRESOURCE

Value: 944

D3D12_MESSAGE_ID_COPYRESOURCE_NULLSRC

Value: 945

D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDSRCRESOURCE

Value: 946

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLDST

Value: 947

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDDSTRESOURCE

Value: 948

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLSRC

Value: 949

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDSRCRESOURCE

Value: 950

D3D12_MESSAGE_ID_PIPELINE_STATE_TYPE_MISMATCH

Value: 951

D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_NOT_SET

Value: 952

D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_MISMATCH

Value: 953

D3D12_MESSAGE_ID_RESOURCE_BARRIER_ZERO_BARRIERS

Value: 954

D3D12_MESSAGE_ID_BEGIN_END_EVENT_MISMATCH

Value: 955

D3D12_MESSAGE_ID_RESOURCE_BARRIER_POSSIBLE_BEFORE_AFTER_MISMATCH

Value: 956

D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_BEGIN_END

Value: 957

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INVALID_RESOURCE

Value: 958

D3D12_MESSAGE_ID_USE_OF_ZERO_REFCOUNT_OBJECT

Value: 959

D3D12_MESSAGE_ID_OBJECT_EVICTED_WHILE_STILL_IN_USE

Value: 960

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_DESCRIPTOR_ACCESS_OUT_OF_BOUNDS

Value: 961

D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_INVALIDLIBRARYBLOB

Value: 962

D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_DRIVERVERSIONMISMATCH

Value: 963

D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_ADAPTERVERSIONMISMATCH

Value: 964

D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_UNSUPPORTED

Value: 965

D3D12_MESSAGE_ID_CREATE_PIPELINELIBRARY

Value: 966

D3D12_MESSAGE_ID_LIVE_PIPELINELIBRARY

Value: 967

D3D12_MESSAGE_ID_DESTROY_PIPELINELIBRARY

Value: 968

D3D12_MESSAGE_ID_STOREPIPELINE_NONAME

Value: 969

D3D12_MESSAGE_ID_STOREPIPELINE_DUPLICATENAME

Value: 970

D3D12_MESSAGE_ID_LOADPIPELINE_NAMENOTFOUND

Value: 971

D3D12_MESSAGE_ID_LOADPIPELINE_INVALIDDESC

Value: 972

D3D12_MESSAGE_ID_PIPELINELIBRARY_SERIALIZE_NOTENOUGHMEMORY

Value: 973

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_PS_OUTPUT_RT_OUTPUT_MISMATCH

Value: 974

D3D12_MESSAGE_ID_SETEVENTONMULTIPLEFENCECOMPLETION_INVALIDFLAGS

Value: 975

D3D12_MESSAGE_ID_CREATE_QUEUE_VIDEO_NOT_SUPPORTED

Value: 976

D3D12_MESSAGE_ID_CREATE_COMMAND_ALLOCATOR_VIDEO_NOT_SUPPORTED

Value: 977

D3D12_MESSAGE_ID_CREATEQUERY_HEAP_VIDEO_DECODE_STATISTICS_NOT_SUPPORTED

Value: 978

D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDLIST

Value: 979

D3D12_MESSAGE_ID_CREATE_VIDEODECODER

Value: 980

D3D12_MESSAGE_ID_CREATE_VIDEODECODESTREAM

Value: 981

D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDLIST

Value: 982

D3D12_MESSAGE_ID_LIVE_VIDEODECODER

Value: 983

D3D12_MESSAGE_ID_LIVE_VIDEODECODESTREAM

Value: 984

D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDLIST

Value: 985

D3D12_MESSAGE_ID_DESTROY_VIDEODECODER

Value: 986

D3D12_MESSAGE_ID_DESTROY_VIDEODECODESTREAM

Value: 987

D3D12_MESSAGE_ID_DECODE_FRAME_INVALID_PARAMETERS

Value: 988

D3D12_MESSAGE_ID_DEPRECATED_API

Value: 989

D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_COMMAND_LIST_TYPE

Value: 990

D3D12_MESSAGE_ID_COMMAND_LIST_DESCRIPTOR_TABLE_NOT_SET

Value: 991

D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_CONSTANT_BUFFER_VIEW_NOT_SET

Value: 992

D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_SHADER_RESOURCE_VIEW_NOT_SET

Value: 993

D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_UNORDERED_ACCESS_VIEW_NOT_SET

Value: 994

D3D12_MESSAGE_ID_DISCARD_INVALID_SUBRESOURCE_RANGE

Value: 995

D3D12_MESSAGE_ID_DISCARD_ONE_SUBRESOURCE_FOR_MIPS_WITH_RECTS

Value: 996

D3D12_MESSAGE_ID_DISCARD_NO_RECTS_FOR_NON_TEXTURE2D

Value: 997

D3D12_MESSAGE_ID_COPY_ON_SAME_SUBRESOURCE

Value: 998

D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PAGEABLE

Value: 999

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UNSUPPORTED

Value: 1000

D3D12_MESSAGE_ID_STATIC_DESCRIPTOR_INVALID_DESCRIPTOR_CHANGE

Value: 1001

D3D12_MESSAGE_ID_DATA_STATIC_DESCRIPTOR_INVALID_DATA_CHANGE

Value: 1002

D3D12_MESSAGE_ID_DATA_STATIC_WHILE_SET_AT_EXECUTE_DESCRIPTOR_INVALID_DATA_CHANGE

Value: 1003

D3D12_MESSAGE_ID_EXECUTE_BUNDLE_STATIC_DESCRIPTOR_DATA_STATIC_NOT_SET

Value: 1004

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_ACCESS_OUT_OF_BOUNDS

Value: 1005

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SAMPLER_MODE_MISMATCH

Value: 1006

D3D12_MESSAGE_ID_CREATE_FENCE_INVALID_FLAGS

Value: 1007

D3D12_MESSAGE_ID_RESOURCE_BARRIER_DUPLICATE_SUBRESOURCE_TRANSITIONS

Value: 1008

D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PRIORITY

Value: 1009

D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_LARGE_NUM_DESCRIPTORS

Value: 1013

D3D12_MESSAGE_ID_BEGIN_EVENT

Value: 1014

D3D12_MESSAGE_ID_END_EVENT

Value: 1015

D3D12_MESSAGE_ID_CREATEDevice_DEBUG_LAYER_STARTUP_OPTIONS

Value: 1016

D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_DEPTHBOUNDSTEST_UNSUPPORTED

Value: 1017

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_DUPLICATE_SUBOBJECT

Value: 1018

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_UNKNOWN_SUBOBJECT

Value: 1019

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_ZERO_SIZE_STREAM

Value: 1020

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_STREAM

Value: 1021

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CANNOT_DEDUCE_TYPE

Value: 1022

D3D12_MESSAGE_ID_COMMAND_LIST_STATIC_DESCRIPTOR_RESOURCE_DIMENSION_MISMATCH

Value: 1023

D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_PRIVILEGE_FOR_GLOBAL_REALTIME

Value: 1024

D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_HARDWARE_SUPPORT_FOR_GLOBAL_REALTIME

Value: 1025

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_ARCHITECTURE

Value: 1026

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DST

Value: 1027

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE_DIMENSION

Value: 1028

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DST_RANGE_OUT_OF_BOUNDS

Value: 1029

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_SRC

Value: 1030

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE_DIMENSION

Value: 1031

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_SRC_RANGE_OUT_OF_BOUNDS

Value: 1032

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_OFFSET_ALIGNMENT

Value: 1033

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_RESOURCES

Value: 1034

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_SUBRESOURCE_RANGES

Value: 1035

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_RESOURCE

Value: 1036

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_SUBRESOURCE_RANGE

Value: 1037

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_SUBRESOURCE_OUT_OF_BOUNDS

Value: 1038

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_RANGE_OUT_OF_BOUNDS

Value: 1039

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_ZERO_DEPENDENCIES

Value: 1040

D3D12_MESSAGE_ID_DEVICE_CREATE_SHARED_HANDLE_INVALIDARG

Value: 1041

D3D12_MESSAGE_ID_DESCRIPTOR_HANDLE_WITH_INVALID_RESOURCE

Value: 1042

D3D12_MESSAGE_ID_SETDEPTHBOUNDS_INVALIDARGS

Value: 1043

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_STATE_IMPRECISE

Value: 1044

D3D12_MESSAGE_ID_COMMAND_LIST_PIPELINE_STATE_NOT_SET

Value: 1045

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Model_Mismatch

Value: 1046

D3D12_MESSAGE_ID_OBJECT_ACCESSED_WHILE_STILL_IN_USE

Value: 1047

D3D12_MESSAGE_ID_PROGRAMMABLE_MSAA_UNSUPPORTED

Value: 1048

D3D12_MESSAGE_ID_SETSAMPLEPOSITIONS_INVALIDARGS

Value: 1049

D3D12_MESSAGE_ID_RESOLVESUBRESOURCEREGION_INVALID_RECT

Value: 1050

D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDQUEUE

Value: 1051

D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDLIST

Value: 1052

D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDQUEUE

Value: 1053

D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDQUEUE

Value: 1054

D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDLIST

Value: 1055

D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDQUEUE

Value: 1056

D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDQUEUE

Value: 1057

D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDLIST

Value: 1058

D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDQUEUE

Value: 1059

D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSOR

Value: 1060

D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSSTREAM

Value: 1061

D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSOR

Value: 1062

D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSSTREAM

Value: 1063

D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSOR

Value: 1064

D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSSTREAM

Value: 1065

D3D12_MESSAGE_ID_PROCESS_FRAME_INVALID_PARAMETERS

Value: 1066

D3D12_MESSAGE_ID_COPY_INVALIDLAYOUT

Value: 1067

D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION

Value: 1068

D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION_POLICY

Value: 1069

D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION

Value: 1070

D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION

Value: 1071

D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION_POLICY

Value: 1072

D3D12_MESSAGE_ID_LIVE_PROTECTED_RESOURCE_SESSION

Value: 1073

D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION

Value: 1074

D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION_POLICY

Value: 1075

D3D12_MESSAGE_ID_DESTROY_PROTECTED_RESOURCE_SESSION

Value: 1076

D3D12_MESSAGE_ID_PROTECTED_RESOURCE_SESSION_UNSUPPORTED

Value: 1077

D3D12_MESSAGE_ID_FENCE_INVALIDOPERATION

Value: 1078

D3D12_MESSAGE_ID_CREATEQUERY_HEAP_COPY_QUEUE_TIMESTAMPS_NOT_SUPPORTED

Value: 1079

D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_DEFERRED
Value: 1080
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMFIRSTUSE
Value: 1081
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMCLEAR
Value: 1082
D3D12_MESSAGE_ID_CREATE_VIDEODECODERHEAP
Value: 1083
D3D12_MESSAGE_ID_LIVE_VIDEODECODERHEAP
Value: 1084
D3D12_MESSAGE_ID_DESTROY_VIDEODECODERHEAP
Value: 1085
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDARG_RETURN
Value: 1086
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_OUTOFMEMORY_RETURN
Value: 1087
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDADDRESS
Value: 1088
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDHANDLE
Value: 1089
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_DEST
Value: 1090
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_MODE
Value: 1091
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_ALIGNMENT
Value: 1092
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_NOT_SUPPORTED
Value: 1093
D3D12_MESSAGE_ID_SETVIEWINSTANCEMASK_INVALIDARGS
Value: 1094
D3D12_MESSAGE_ID_VIEW_INSTANCING_UNSUPPORTED
Value: 1095
D3D12_MESSAGE_ID_VIEW_INSTANCING_INVALIDARGS
Value: 1096
D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_DECODE_REFERENCE_ONLY_FLAG
Value: 1097

D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_DECODE_REFERENCE_ONLY_FLAG

Value: 1098

D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_FAILURE

Value: 1099

D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_UNSUPPORTED

Value: 1100

D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_INVALID_ID_INPUT

Value: 1101

D3D12_MESSAGE_ID_CREATE_VIDEO_DECODER_UNSUPPORTED

Value: 1102

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Metadata_Error

Value: 1103

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_View_Instancing_Vertex_Size_Exceeded

Value: 1104

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Runtime_Internal_Error

Value: 1105

D3D12_MESSAGE_ID_NO_VIDEO_API_SUPPORT

Value: 1106

D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_INVALID_ID_INPUT

Value: 1107

D3D12_MESSAGE_ID_CREATE_VIDEO_PROCESSOR_CAPS_FAILURE

Value: 1108

D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_UNSUPPORTED_FORMAT

Value: 1109

D3D12_MESSAGE_ID_VIDEO_DECODE_FRAME_INVALID_ARGUMENT

Value: 1110

D3D12_MESSAGE_ID_ENQUEUE_MAKE_RESIDENT_INVALID_FLAGS

Value: 1111

D3D12_MESSAGE_ID_OPENEXISTINGHEAP_UNSUPPORTED

Value: 1112

D3D12_MESSAGE_ID_VIDEO_PROCESS_FRAMES_INVALID_ARGUMENT

Value: 1113

D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_UNSUPPORTED

Value: 1114

D3D12_MESSAGE_ID_CREATE_COMMANDRECODER

Value: 1115

D3D12_MESSAGE_ID_LIVE_COMMANDRECORDER
Value: 1116
D3D12_MESSAGE_ID_DESTROY_COMMANDRECORDER
Value: 1117
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_VIDEO_NOT_SUPPORTED
Value: 1118
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_SUPPORT_FLAGS
Value: 1119
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_FLAGS
Value: 1120
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_MORE_RECORDERS_THAN_LOGICAL_PROCESSORS
Value: 1121
D3D12_MESSAGE_ID_CREATE_COMMANDPOOL
Value: 1122
D3D12_MESSAGE_ID_LIVE_COMMANDPOOL
Value: 1123
D3D12_MESSAGE_ID_DESTROY_COMMANDPOOL
Value: 1124
D3D12_MESSAGE_ID_CREATE_COMMAND_POOL_INVALID_FLAGS
Value: 1125
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_VIDEO_NOT_SUPPORTED
Value: 1126
D3D12_MESSAGE_ID_COMMAND_RECORDER_SUPPORT_FLAGS_MISMATCH
Value: 1127
D3D12_MESSAGE_ID_COMMAND_RECORDER_CONTENTION
Value: 1128
D3D12_MESSAGE_ID_COMMAND_RECORDER_USAGE_WITH_CREATECOMMANDLIST_COMMAND_LIST
Value: 1129
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_USAGE_WITH_CREATECOMMANDLIST1_COMMAND_LIST
Value: 1130
D3D12_MESSAGE_ID_CANNOT_EXECUTE_EMPTY_COMMAND_LIST
Value: 1131
D3D12_MESSAGE_ID_CANNOT_RESET_COMMAND_POOL_WITH_OPEN_COMMAND_LISTS
Value: 1132
D3D12_MESSAGE_ID_CANNOT_USE_COMMAND_RECORDER_WITHOUT_CURRENT_TARGET
Value: 1133

D3D12_MESSAGE_ID_CANNOT_CHANGE_COMMAND_RECORDER_TARGET_WHILE_RECORDING

Value: 1134

D3D12_MESSAGE_ID_COMMAND_POOL_SYNC

Value: 1135

D3D12_MESSAGE_ID_EVICT_UNDERFLOW

Value: 1136

D3D12_MESSAGE_ID_CREATE_META_COMMAND

Value: 1137

D3D12_MESSAGE_ID_LIVE_META_COMMAND

Value: 1138

D3D12_MESSAGE_ID_DESTROY_META_COMMAND

Value: 1139

D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_DST_RESOURCE

Value: 1140

D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_SRC_RESOURCE

Value: 1141

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE

Value: 1142

D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE

Value: 1143

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_BUFFER

Value: 1144

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_RESOURCE_DESC

Value: 1145

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_UNSUPPORTED

Value: 1146

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_DIMENSION

Value: 1147

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_FLAGS

Value: 1148

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_OFFSET

Value: 1149

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_DIMENSION

Value: 1150

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_FLAGS

Value: 1151

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_OUTOFMEMORY_RETURN

Value: 1152

D3D12_MESSAGE_ID_CANNOT_CREATE_GRAPHICS_AND_VIDEO_COMMAND_RECORDER

Value: 1153

D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_POSSIBLY_MISMATCHING_PROPERTIES

Value: 1154

D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE

Value: 1155

D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INCOMPATIBLE_WITH_STRUCTURED_BUFFERS

Value: 1156

D3D12_MESSAGE_ID_COMPUTE_ONLY_DEVICE_OPERATION_UNSUPPORTED

Value: 1157

D3D12_MESSAGE_ID_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INVALID

Value: 1158

D3D12_MESSAGE_ID_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_INVALID

Value: 1159

D3D12_MESSAGE_ID_COPY_RAYTRACING_ACCELERATION_STRUCTURE_INVALID

Value: 1160

D3D12_MESSAGE_ID_DISPATCH_RAYS_INVALID

Value: 1161

D3D12_MESSAGE_ID_GET_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO_INVALID

Value: 1162

D3D12_MESSAGE_ID_CREATE_LIFETIMETRACKER

Value: 1163

D3D12_MESSAGE_ID_LIVE_LIFETIMETRACKER

Value: 1164

D3D12_MESSAGE_ID_DESTROY_LIFETIMETRACKER

Value: 1165

D3D12_MESSAGE_ID_DESTROYOWNEDOBJECT_OBJECTNOTOWNED

Value: 1166

D3D12_MESSAGE_ID_CREATE_TRACKEDWORKLOAD

Value: 1167

D3D12_MESSAGE_ID_LIVE_TRACKEDWORKLOAD

Value: 1168

D3D12_MESSAGE_ID_DESTROY_TRACKEDWORKLOAD

Value: 1169

D3D12_MESSAGE_ID_RENDER_PASS_ERROR

Value: 1170

D3D12_MESSAGE_ID_META_COMMAND_ID_INVALID

Value: 1171

D3D12_MESSAGE_ID_META_COMMAND_UNSUPPORTED_PARAMS

Value: 1172

D3D12_MESSAGE_ID_META_COMMAND_FAILED_ENUMERATION

Value: 1173

D3D12_MESSAGE_ID_META_COMMAND_PARAMETER_SIZE_MISMATCH

Value: 1174

D3D12_MESSAGE_ID_UNINITIALIZED_META_COMMAND

Value: 1175

D3D12_MESSAGE_ID_META_COMMAND_INVALID_GPU_VIRTUAL_ADDRESS

Value: 1176

D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDLIST

Value: 1177

D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDLIST

Value: 1178

D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDLIST

Value: 1179

D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDQUEUE

Value: 1180

D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDQUEUE

Value: 1181

D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDQUEUE

Value: 1182

D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONESTIMATOR

Value: 1183

D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONESTIMATOR

Value: 1184

D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONESTIMATOR

Value: 1185

D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONVECTORHEAP

Value: 1186

D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONVECTORHEAP

Value: 1187

D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONVECTORHEAP

Value: 1188

D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOADS

Value: 1189

D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOAD_PAIRS

Value: 1190

D3D12_MESSAGE_ID_OUT_OF_ORDER_TRACKED_WORKLOAD_PAIR

Value: 1191

D3D12_MESSAGE_ID_CANNOT_ADD_TRACKED_WORKLOAD

Value: 1192

D3D12_MESSAGE_ID_INCOMPLETE_TRACKED_WORKLOAD_PAIR

Value: 1193

D3D12_MESSAGE_ID_CREATE_STATE_OBJECT_ERROR

Value: 1194

D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_ERROR

Value: 1195

D3D12_MESSAGE_ID_GET_SHADER_STACK_SIZE_ERROR

Value: 1196

D3D12_MESSAGE_ID_GET_PIPELINE_STACK_SIZE_ERROR

Value: 1197

D3D12_MESSAGE_ID_SET_PIPELINE_STACK_SIZE_ERROR

Value: 1198

D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_SIZE_INVALID

Value: 1199

D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_INVALID

Value: 1200

D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_DRIVER_REPORTED_ISSUE

Value: 1201

D3D12_MESSAGE_ID_RENDER_PASS_INVALID_RESOURCE_BARRIER

Value: 1202

D3D12_MESSAGE_ID_RENDER_PASS_DISALLOWED_API_CALLED

Value: 1203

D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_NEST_RENDER_PASSES

Value: 1204

D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_END_WITHOUT_BEGIN

Value: 1205

D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_CLOSE_COMMAND_LIST

Value: 1206

D3D12_MESSAGE_ID_RENDER_PASS_GPU_WORK_WHILE_SUSPENDED

Value: 1207

D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_SUSPEND_RESUME

Value: 1208

D3D12_MESSAGE_ID_RENDER_PASS_NO_PRIOR_SUSPEND_WITHIN_EXECUTECOMMANDLISTS

Value: 1209

D3D12_MESSAGE_ID_RENDER_PASS_NO_SUBSEQUENT_RESUME_WITHIN_EXECUTECOMMANDLISTS

Value: 1210

D3D12_MESSAGE_ID_TRACKED_WORKLOAD_COMMAND_QUEUE_MISMATCH

Value: 1211

D3D12_MESSAGE_ID_TRACKED_WORKLOAD_NOT_SUPPORTED

Value: 1212

D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_NO_ACCESS

Value: 1213

D3D12_MESSAGE_ID_RENDER_PASS_UNSUPPORTED_RESOLVE

Value: 1214

D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INVALID_RESOURCE_PTR

Value: 1215

D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_SIGNAL

Value: 1216

D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_WAIT

Value: 1217

D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_ESTIMATOR_INVALID_ARGUMENT

Value: 1218

D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT

Value: 1219

D3D12_MESSAGE_ID_ESTIMATE_MOTION_INVALID_ARGUMENT

Value: 1220

D3D12_MESSAGE_ID_RESOLVE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT

Value: 1221

D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_HEAP_TYPE

Value: 1222

D3D12_MESSAGE_ID_SET_BACKGROUND_PROCESSING_MODE_INVALID_ARGUMENT

Value: 1223

D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE_FOR_FEATURE_LEVEL

Value: 1224

D3D12_MESSAGE_ID_CREATE_VIDEOEXTENSIONCOMMAND

Value: 1225

D3D12_MESSAGE_ID_LIVE_VIDEOEXTENSIONCOMMAND

Value: 1226

D3D12_MESSAGE_ID_DESTROY_VIDEOEXTENSIONCOMMAND

Value: 1227

D3D12_MESSAGE_ID_INVALID_VIDEO_EXTENSION_COMMAND_ID

Value: 1228

D3D12_MESSAGE_ID_VIDEO_EXTENSION_COMMAND_INVALID_ARGUMENT

Value: 1229

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_NOT_UNIQUE_IN_DXIL_LIBRARY

Value: 1230

D3D12_MESSAGE_ID_VARIABLE_SHADING_RATE_NOT_ALLOWED_WITH_TIR

Value: 1231

D3D12_MESSAGE_ID_GEOMETRY_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE

Value: 1232

D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_SHADING_RATE

Value: 1233

D3D12_MESSAGE_ID_RSSETSHADING_RATE_SHADING_RATE_NOT_PERMITTED_BY_CAP

Value: 1234

D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_COMBINER

Value: 1235

D3D12_MESSAGE_ID_RSSETSHADINGRATEIMAGE_REQUIRES_TIER_2

Value: 1236

D3D12_MESSAGE_ID_RSSETSHADINGRATE_REQUIRES_TIER_1

Value: 1237

D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_FORMAT

Value: 1238

D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_ARRAY_SIZE

Value: 1239

D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_MIP_LEVEL

Value: 1240

D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_COUNT

Value: 1241

D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_QUALITY

Value: 1242

D3D12_MESSAGE_ID_NON_RETAIL_SHADER_MODEL_WONT_VALIDATE

Value: 1243

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_AS_ROOT_SIGNATURE_MISMATCH

Value: 1244

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_ROOT_SIGNATURE_MISMATCH

Value: 1245

D3D12_MESSAGE_ID_ADD_TO_STATE_OBJECT_ERROR

Value: 1246

D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION_INVALID_ARGUMENT

Value: 1247

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_PSO_DESC_MISMATCH

Value: 1248

D3D12_MESSAGE_ID_CREATEPIPELINESTATE_MS_INCOMPLETE_TYPE

Value: 1249

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_AS_NOT_MS_MISMATCH

Value: 1250

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_MS_NOT_PS_MISMATCH

Value: 1251

D3D12_MESSAGE_ID_NONZERO_SAMPLER_FEEDBACK_MIP_REGION_WITH_INCOMPATIBLE_FORMAT

Value: 1252

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_INPUT_LAYOUT_SHADER_MISMATCH

Value: 1253

D3D12_MESSAGE_ID_EMPTY_DISPATCH

Value: 1254

D3D12_MESSAGE_ID_RESOURCE_FORMATQUIRES_SAMPLER_FEEDBACK_CAPABILITY

Value: 1255

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_MIP_REGION

Value: 1256

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_DIMENSION

Value: 1257

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_COUNT

Value: 1258

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_QUALITY

Value: 1259

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_LAYOUT

Value: 1260

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAPQUIRES_UNORDERED_ACCESS_FLAG

Value: 1261

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_NULL_ARGUMENTS

Value: 1262

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_UAV_REQUIRES_SAMPLER_FEEDBACK_CAPABILITY

Value: 1263

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_REQUIRES_FEEDBACK_MAP_FORMAT

Value: 1264

D3D12_MESSAGE_ID_CREATEMESHSHADER_INVALIDSHADERBYTECODE

Value: 1265

D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTOFMEMORY

Value: 1266

D3D12_MESSAGE_ID_CREATEMESHSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE

Value: 1267

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_INVALID_FORMAT

Value: 1268

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_INVALID_MIP_LEVEL_COUNT

Value: 1269

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_ARRAY_SIZE_MISMATCH

Value: 1270

D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_MISMATCHING_TARGETED_RESOURCE

Value: 1271

D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTPUTEXCEEDSMAXSIZE

Value: 1272

D3D12_MESSAGE_ID_CREATEMESHSHADER_GROUPSHAREDEXCEEDSMAXSIZE

Value: 1273

D3D12_MESSAGE_ID_VERTEX_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE

Value: 1274

D3D12_MESSAGE_ID_MESH_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE

Value: 1275

D3D12_MESSAGE_ID_CREATEMESHSHADER_MISMATCHEDASMSPAYLOADSIZE

Value: 1276

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_UNBOUNDED_STATIC_DESCRIPTOR

Value: 1277

D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_INVALIDSHADERBYTECODE

Value: 1278

D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_OUTOFMEMORY

Value: 1279

D3D12_MESSAGE_ID_CREATE_SHADERCACHESESSION

Value: 1280

D3D12_MESSAGE_ID_LIVE_SHADERCACHESESSION

Value: 1281

D3D12_MESSAGE_ID_DESTROY_SHADERCACHESESSION

Value: 1282

D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_INVALIDARGS

Value: 1283

D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_DISABLED

Value: 1284

D3D12_MESSAGE_ID_CREATESHADERCACHESESSION_ALREADYOPEN

Value: 1285

D3D12_MESSAGE_ID_SHADERCACHECONTROL_DEVELOPERMODE

Value: 1286

D3D12_MESSAGE_ID_SHADERCACHECONTROL_INVALIDFLAGS

Value: 1287

D3D12_MESSAGE_ID_SHADERCACHECONTROL_STATEALREADYSET

Value: 1288

D3D12_MESSAGE_ID_SHADERCACHECONTROL_IGNOREDFLAG

Value: 1289

D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_ALREADYPRESENT

Value: 1290

D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_HASHCOLLISION

Value: 1291

D3D12_MESSAGE_ID_SHADERCACHESESSION_STOREVALUE_CACHEFULL

Value: 1292

D3D12_MESSAGE_ID_SHADERCACHESESSION_FINDVALUE_NOTFOUND

Value: 1293

D3D12_MESSAGE_ID_SHADERCACHESESSION_CORRUPT

Value: 1294

D3D12_MESSAGE_ID_SHADERCACHESESSION_DISABLED

Value: 1295

D3D12_MESSAGE_ID_OVERSIZED_DISPATCH

Value: 1296

D3D12_MESSAGE_ID_CREATE_VIDEOENCODER

Value: 1297

D3D12_MESSAGE_ID_LIVE_VIDEOENCODER

Value: 1298

D3D12_MESSAGE_ID_DESTROY_VIDEOENCODER

Value: 1299

D3D12_MESSAGE_ID_CREATE_VIDEOENCODERHEAP

Value: 1300

D3D12_MESSAGE_ID_LIVE_VIDEOENCODERHEAP

Value: 1301

D3D12_MESSAGE_ID_DESTROY_VIDEOENCODERHEAP

Value: 1302

D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_ENCODE_REFERENCE_ONLY_FLAG

Value: 1303

D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_ENCODE_REFERENCE_ONLY_FLAG

Value: 1304

D3D12_MESSAGE_ID_ENCODE_FRAME_INVALID_PARAMETERS

Value: 1305

D3D12_MESSAGE_ID_ENCODE_FRAME_UNSUPPORTED_PARAMETERS

Value: 1306

D3D12_MESSAGE_ID_RESOLVE_ENCODER_OUTPUT_METADATA_INVALID_PARAMETERS

Value: 1307

D3D12_MESSAGE_ID_RESOLVE_ENCODER_OUTPUT_METADATA_UNSUPPORTED_PARAMETERS

Value: 1308

D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_INVALID_PARAMETERS

Value: 1309

D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_UNSUPPORTED_PARAMETERS

Value: 1310

D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_HEAP_INVALID_PARAMETERS

Value: 1311

D3D12_MESSAGE_ID_CREATE_VIDEO_ENCODER_HEAP_UNSUPPORTED_PARAMETERS

Value: 1312

D3D12_MESSAGE_ID_D3D12_MESSAGES_END

Remarks

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

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_MESSAGE_SEVERITY enumeration (d3d12sdklayers.h)

Article 02/22/2024

Debug message severity levels for an information queue.

Syntax

C++

```
typedef enum D3D12_MESSAGE_SEVERITY {
    D3D12_MESSAGE_SEVERITY_CORRUPTION = 0,
    D3D12_MESSAGE_SEVERITY_ERROR,
    D3D12_MESSAGE_SEVERITY_WARNING,
    D3D12_MESSAGE_SEVERITY_INFO,
    D3D12_MESSAGE_SEVERITY_MESSAGE
} ;
```

Constants

[+] Expand table

<code>D3D12_MESSAGE_SEVERITY_CORRUPTION</code> Value: 0 Indicates a corruption error.
<code>D3D12_MESSAGE_SEVERITY_ERROR</code> Indicates an error.
<code>D3D12_MESSAGE_SEVERITY_WARNING</code> Indicates a warning.
<code>D3D12_MESSAGE_SEVERITY_INFO</code> Indicates an information message.
<code>D3D12_MESSAGE_SEVERITY_MESSAGE</code> Indicates a message other than corruption, error, warning or information.

Remarks

Use these values to allow or deny message categories to pass through the storage and retrieval filters for an information queue (see [D3D12_INFO_QUEUE_FILTER](#)). This API is used by [AddApplicationMessage](#) and [AddMessage](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_RLDO_FLAGS enumeration (d3d12sdklayers.h)

Article02/22/2024

Specifies options for the amount of information to report about a live device object's lifetime.

Syntax

C++

```
typedef enum D3D12_RLDO_FLAGS {
    D3D12_RLDO_NONE = 0,
    D3D12_RLDO_SUMMARY = 0x1,
    D3D12_RLDO_DETAIL = 0x2,
    D3D12_RLDO_IGNORE_INTERNAL = 0x4
};
```

Constants

[] Expand table

D3D12_RLDO_NONE Value: <i>0</i>
D3D12_RLDO_SUMMARY Value: <i>0x1</i> Obtain a summary about a live device object's lifetime.
D3D12_RLDO_DETAIL Value: <i>0x2</i> Obtain detailed information about a live device object's lifetime.
D3D12_RLDO_IGNORE_INTERNAL Value: <i>0x4</i> 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.

Remarks

This enumeration is used by [ID3D12DebugDevice::ReportLiveDeviceObjects](#).

Requirements

 [Expand table](#)

Requirement	Value
Header	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug interface (d3d12sdklayers.h)

Article 02/22/2024

An interface used to turn on the debug layer. See [EnableDebugLayer](#) for more information.

Inheritance

The **ID3D12Debug** interface inherits from the [IUnknown](#) interface. **ID3D12Debug** also has these types of members:

Methods

The **ID3D12Debug** interface has these methods.

[+] Expand table

<p>ID3D12Debug::EnableDebugLayer</p> <p>Enables the debug layer. (<code>ID3D12Debug.EnableDebugLayer</code>)</p>

Remarks

This interface is obtained by querying it from [D3D12GetDebugInterface](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug::EnableDebugLayer method (d3d12sdklayers.h)

Article 02/22/2024

Enables the debug layer.

Syntax

C++

```
void EnableDebugLayer();
```

Return value

None

Remarks

To enable the debug layers using this API, it must be called before the D3D12 device is created. Calling this API after creating the D3D12 device will cause the D3D12 runtime to remove the device.

Examples

Enable the D3D12 debug layer.

C++

```
// Enable the D3D12 debug layer.
{
    ComPtr<ID3D12Debug> debugController;
    if (SUCCEEDED(D3D12GetDebugInterface(IID_PPV_ARGS(&debugController))))
    {
        debugController->EnableDebugLayer();
    }
}
```

Refer to the [Example Code in the D3D12 Reference](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug1 interface (d3d12sdklayers.h)

Article02/22/2024

Adds GPU-Based Validation and Dependent Command Queue Synchronization to the debug layer.

Inheritance

The **ID3D12Debug1** interface inherits from the [IUnknown](#) interface. **ID3D12Debug1** also has these types of members:

Methods

The **ID3D12Debug1** interface has these methods.

[] [Expand table](#)

ID3D12Debug1::EnableDebugLayer
Enables the debug layer. (<code>ID3D12Debug1.EnableDebugLayer</code>)
ID3D12Debug1::SetEnableGPUBasedValidation
This method enables or disables GPU-Based Validation (GBV) before creating a device with the debug layer enabled.
ID3D12Debug1::SetEnableSynchronizedCommandQueueValidation
Enables or disables dependent command queue synchronization when using a D3D12 device with the debug layer enabled.

Remarks

This interface is currently in Preview mode.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug1::EnableDebugLayer method (d3d12sdklayers.h)

Article 02/22/2024

Enables the debug layer.

Syntax

C++

```
void EnableDebugLayer();
```

Return value

None

Remarks

This method is identical to [ID3D12Debug::EnableDebugLayer](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12Debug1::SetEnableGPUBasedValidation method (d3d12sdklayers.h)

Article 02/22/2024

This method enables or disables GPU-Based Validation (GBV) before creating a device with the debug layer enabled.

Syntax

C++

```
void SetEnableGPUBasedValidation(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: **BOOL**

TRUE to enable GPU-Based Validation, otherwise FALSE.

Return value

None

Remarks

GPU-Based Validation can only be enabled/disabled prior to creating a device. By default, GPU-Based Validation is disabled. To disable GPU-Based Validation after initially enabling it the device must be fully released and recreated; disabling or enabling it after device creation will cause device removal.

For more information, see [Using D3D12 Debug Layer GPU-Based Validation](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug1::SetEnableSynchronizedCommandQueueValidation method (d3d12sdklayers.h)

Article 02/22/2024

Enables or disables dependent command queue synchronization when using a D3D12 device with the debug layer enabled.

Syntax

C++

```
void SetEnableSynchronizedCommandQueueValidation(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: **BOOL**

TRUE to enable Dependent Command Queue Synchronization, otherwise FALSE.

Return value

None

Remarks

Dependent Command Queue Synchronization is a D3D12 Debug Layer feature that gives the debug layer the ability to track resource states more accurately when enabled. Dependent Command Queue Synchronization is enabled by default.

When Dependent Command Queue Synchronization is enabled, the debug layer holds back actual submission of GPU work until all outstanding fence [Wait](#) conditions are met. This gives the debug layer the ability to make reasonable assumptions about GPU state

(such as resource states) on the CPU-timeline when multiple command queues are potentially doing concurrent work.

With Dependent Command Queue Synchronization disabled, all resource states tracked by the debug layer are cleared each time [ID3D12CommandQueue::Signal](#) is called. This results in significantly less useful resource state validation.

Disabling Dependent Command Queue Synchronization may reduce some debug layer performance overhead when using multiple command queues. However, it is suggested to leave it enabled unless this overhead is problematic. Note that applications that use only a single command queue will see no performance changes with Dependent Command Queue Synchronization disabled.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug2 interface (d3d12sdklayers.h)

Article02/22/2024

Adds configurable levels of GPU-based validation to the debug layer.

Inheritance

The **ID3D12Debug2** interface inherits from the [IUnknown](#) interface.

Methods

The **ID3D12Debug2** interface has these methods.

[+] [Expand table](#)

[ID3D12Debug2::SetGPUBasedValidationFlags](#)

This method configures the level of GPU-based validation that the debug device is to perform at runtime. (`ID3D12Debug2::SetGPUBasedValidationFlags`)

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#) [IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug2::SetGPUBasedValidationFlags method (d3d12sdklayers.h)

Article 02/22/2024

This method configures the level of GPU-based validation that the debug device is to perform at runtime.

Syntax

C++

```
void SetGPUBasedValidationFlags(  
    D3D12_GPU_BASED_VALIDATION_FLAGS Flags  
)
```

Parameters

Flags

Type: [D3D12_GPU_BASED_VALIDATION_FLAGS](#)

Specifies the level of GPU-based validation to perform at runtime.

Return value

None

Remarks

This method overrides the default behavior of GPU-based validation so it must be called before creating the Direct3D 12 device. These settings can't be changed or cancelled after the device is created. If you want to change the behavior of GPU-based validation at a later time, the device must be destroyed and recreated with different parameters.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug2](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug3 interface (d3d12sdklayers.h)

Article02/22/2024

Adds configurable levels of GPU-based validation to the debug layer.

Inheritance

The [ID3D12Debug3](#) interface inherits from the [ID3D12Debug](#) interface.

Methods

The [ID3D12Debug3](#) interface has these methods.

[+] [Expand table](#)

ID3D12Debug3::SetEnableGPUBasedValidation
This method enables or disables GPU-based validation (GBV) before creating a device with the debug layer enabled.
ID3D12Debug3::SetEnableSynchronizedCommandQueueValidation
Enables or disables dependent command queue synchronization when using a Direct3D 12 device with the debug layer enabled.
ID3D12Debug3::SetGPUBasedValidationFlags
This method configures the level of GPU-based validation that the debug device is to perform at runtime. (<code>ID3D12Debug3::SetGPUBasedValidationFlags</code>)

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug3::SetEnableGPUBasedValidation method (d3d12sdklayers.h)

Article 02/22/2024

This method enables or disables GPU-based validation (GBV) before creating a device with the debug layer enabled.

Syntax

C++

```
void SetEnableGPUBasedValidation(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: **BOOL**

TRUE to enable GPU-based validation, otherwise FALSE.

Return value

None

Remarks

GPU-based validation can be enabled/disabled only prior to creating a device. By default, GPU-based validation is disabled. To disable GPU-based validation after initially enabling it, the device must be fully released and recreated.

For more information, see [Using D3D12 Debug Layer GPU-based validation](#).

Requirements

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12Debug3::SetEnableSynchronizedCommandQueueValidation method (d3d12sdklayers.h)

Article 02/22/2024

Enables or disables dependent command queue synchronization when using a Direct3D 12 device with the debug layer enabled.

Syntax

C++

```
void SetEnableSynchronizedCommandQueueValidation(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: **BOOL**

TRUE to enable Dependent Command Queue Synchronization, otherwise FALSE.

Return value

None

Remarks

Dependent Command Queue Synchronization is a D3D12 Debug Layer feature that gives the debug layer the ability to track resource states more accurately when enabled. Dependent Command Queue Synchronization is enabled by default.

When Dependent Command Queue Synchronization is enabled, the debug layer holds back actual submission of GPU work until all outstanding fence [Wait](#) conditions are met. This gives the debug layer the ability to make reasonable assumptions about GPU state

(such as resource states) on the CPU-timeline when multiple command queues are potentially doing concurrent work.

With Dependent Command Queue Synchronization disabled, all resource states tracked by the debug layer are cleared each time [ID3D12CommandQueue::Signal](#) is called. This results in significantly less useful resource state validation.

Disabling Dependent Command Queue Synchronization may reduce some debug layer performance overhead when using multiple command queues. However, it is suggested to leave it enabled unless this overhead is problematic. Note that applications that use only a single command queue will see no performance changes with Dependent Command Queue Synchronization disabled.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug3::SetGPUBasedValidationFlags method (d3d12sdklayers.h)

Article 02/22/2024

This method configures the level of GPU-based validation that the debug device is to perform at runtime.

Syntax

C++

```
void SetGPUBasedValidationFlags(  
    D3D12_GPU_BASED_VALIDATION_FLAGS Flags  
)
```

Parameters

Flags

Type: [D3D12_GPU_BASED_VALIDATION_FLAGS](#)

Specifies the level of GPU-based validation to perform at runtime.

Return value

None

Remarks

This method overrides the default behavior of GPU-based validation so it must be called before creating the D3D12 Device. These settings can't be changed or cancelled after the device is created. If you want to change the behavior of GPU-based validation at a later time, the device must be destroyed and recreated with different parameters.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug4 interface (d3d12sdklayers.h)

Article 02/22/2024

Adds the ability to disable the debug layer.

Inheritance

The **ID3D12Debug4** interface inherits from the **ID3D12Debug3** interface.

Methods

The **ID3D12Debug4** interface has these methods.

[+] Expand table

<p>ID3D12Debug4::DisableDebugLayer</p> <p>Disables the debug layer.</p>

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

- [Debug layer interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug4::DisableDebugLayer method (d3d12sdklayers.h)

Article 02/22/2024

Disables the debug layer.

`DisableDebugLayer` takes effect only when all [ID3D12Device](#) objects (for a given adapter) have been released. See [Remarks](#) for [D3D12CreateDevice](#).

Syntax

C++

```
void DisableDebugLayer();
```

Return value

None

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

- [ID3D12Debug4](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug5 interface (d3d12sdklayers.h)

Article 02/22/2024

Adds to the debug layer the ability to configure the auto-naming of objects.

Inheritance

The ID3D12Debug5 interface inherits from the ID3D12Debug4 interface.

Methods

The ID3D12Debug5 interface has these methods.

[+] Expand table

ID3D12Debug5::SetEnableAutoName	
	Configures the auto-naming of objects.

Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

- [Debug layer interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12Debug5::SetEnableAutoName method (d3d12sdklayers.h)

Article 02/22/2024

Configures the auto-naming of objects.

Syntax

C++

```
void SetEnableAutoName(  
    BOOL Enable  
) ;
```

Parameters

Enable

Type: [BOOL](#)

`true` to enable auto-naming; `false` to disable auto-naming.

Return value

None

Remarks

By default, objects are not named unless you use [ID3D12Object::SetName](#) or [ID3D12Object::SetPrivateData](#) to assign a name.

It's a best practice to name all of your Direct3D 12 objects; at least in debug builds. Failing that, you might find it convenient to allow automatic name assignment in order to cover the gaps. Direct3D 12 objects created with auto-name enabled are automatically assigned a name, which is used for debug layer output and for DRED page fault data.

So as not to create a dependency on a specific auto-naming format, you can't retrieve the auto-name strings by using [ID3D12Object::GetName](#) or

[ID3D12Object::GetPrivateData](#). But, to generate a unique name string, Direct3D 12 uses the locally-unique identifier (LUID) assigned to every **ID3D12DeviceChild** object at create time. You can retrieve that LUID by using [ID3D12Object::GetPrivateData](#) with the **REFGUID** value *WKPDID_D3D12UniqueObjectId*. You might find that useful for your own object-naming schemas.

When debugging existing software, you can control auto-naming by using the **D3DConfig** graphics tools utility and the command `d3dconfig.exe device auto-debug-name=forced-on`.

Any object given a name using [ID3D12Object::SetName](#) or [ID3D12Object::SetPrivateData](#) uses the assigned name instead of the auto-name.

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10 Build 20348
Minimum supported server	Windows 10 Build 20348
Target Platform	Windows
Header	d3d12sdklayers.h

See also

- [ID3D12Debug5](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12Debug6 interface (d3d12sdklayers.h)

Article 02/22/2024

A debug interface controls debug settings.

Requires the DirectX 12 Agility SDK 1.7 or later.

Inheritance

The ID3D12Debug6 interface inherits from the ID3D12Debug5 interface.

Methods

The ID3D12Debug6 interface has these methods.

[+] Expand table

ID3D12Debug6::SetForceLegacyBarrierValidation
TBD

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

Feedback

Was this page helpful?

 Yes

 No

ID3D12Debug6::SetForceLegacyBarrierValidation method (d3d12sdklayers.h)

Article 02/22/2024

TBD

Requires the DirectX 12 Agility SDK 1.7 or later.

Syntax

C++

```
void SetForceLegacyBarrierValidation(  
    BOOL Enable  
) ;
```

Parameters

Enable

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12DebugCommandList interface (d3d12sdklayers.h)

Article 02/22/2024

Provides methods to monitor and debug a command list.

Inheritance

The **ID3D12DebugCommandList** interface inherits from the [IUnknown](#) interface.

ID3D12DebugCommandList also has these types of members:

Methods

The **ID3D12DebugCommandList** interface has these methods.

 [Expand table](#)

ID3D12DebugCommandList::AssertResourceState
Checks whether a resource, or subresource, is in a specified state, or not. (ID3D12DebugCommandList::AssertResourceState)
ID3D12DebugCommandList::GetFeatureMask
Returns the debug feature flags that have been set on a command list.
ID3D12DebugCommandList::SetFeatureMask
Turns the debug features for a command list on or off.

Requirements

 [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList::AssertResourceState method (d3d12sdklayers.h)

Article 02/22/2024

Checks whether a resource, or subresource, is in a specified state, or not.

Syntax

C++

```
BOOL AssertResourceState(  
    [in] ID3D12Resource *pResource,  
    UINT Subresource,  
    UINT State  
)
```

Parameters

[in] pResource

Type: [ID3D12Resource*](#)

Specifies the [ID3D12Resource](#) to check.

Subresource

Type: [UINT](#)

The index of the subresource to check. This can be set to an index, or D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES.

State

Type: [UINT](#)

Specifies the state to check for. This can be one or more D3D12_RESOURCE_STATES flags Or'ed together.

Return value

Type: [BOOL](#)

This method returns true if the resource or subresource is in the specified state, false otherwise.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList::GetFeatureMask method (d3d12sdklayers.h)

Article02/22/2024

Returns the debug feature flags that have been set on a command list.

Syntax

C++

```
D3D12_DEBUG_FEATURE GetFeatureMask();
```

Return value

Type: [D3D12_DEBUG_FEATURE](#)

A bit mask containing the set debug features.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList::SetFeatureMask method (d3d12sdklayers.h)

Article 02/22/2024

Turns the debug features for a command list on or off.

Syntax

C++

```
HRESULT SetFeatureMask(  
    D3D12_DEBUG_FEATURE Mask  
);
```

Parameters

Mask

Type: [D3D12_DEBUG_FEATURE](#)

A combination of feature-mask flags that are combined by using a bitwise OR operation. If a flag is present, that feature will be set to on, otherwise the feature will be set to off.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList1 interface (d3d12sdklayers.h)

Article 07/22/2021

This interface enables modification of additional command list debug layer settings.

Inheritance

The **ID3D12DebugCommandList1** interface inherits from the [IUnknown](#) interface.

ID3D12DebugCommandList1 also has these types of members:

Methods

The **ID3D12DebugCommandList1** interface has these methods.

[+] Expand table

ID3D12DebugCommandList1::AssertResourceState
Validates that the given state matches the state of the subresource, assuming the state of the given subresource is known during recording of a command list (e.g.
ID3D12DebugCommandList1::GetDebugParameter
Gets optional Command List Debug Layer settings.
ID3D12DebugCommandList1::SetDebugParameter
Modifies optional Debug Layer settings of a command list.

Remarks

This interface is currently in Preview mode.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList1::AssertResourceState method (d3d12sdklayers.h)

Article 02/22/2024

Validates that the given state matches the state of the subresource, assuming the state of the given subresource is known during recording of a command list (e.g. the resource was transitioned earlier in the same command list recording). If the state is not yet known this method sets the known state for further validation later in the same command list recording.

Syntax

C++

```
BOOL AssertResourceState(  
    [in] ID3D12Resource *pResource,  
    UINT             Subresource,  
    UINT             State  
) ;
```

Parameters

[in] pResource

Type: [ID3D12Resource*](#)

Specifies the [ID3D12Resource](#) to check.

Subresource

Type: [UINT](#)

The index of the subresource to check. This can be set to an index, or [D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES](#).

State

Type: [UINT](#)

Specifies the state to check for. This can be one or more [D3D12_RESOURCE_STATES](#) flags Or'ed together.

Return value

Type: **BOOL**

This method returns **true** if the tracked state of the resource or subresource matches the specified state, **false** otherwise.

Remarks

Since execution of command lists occurs sometime after recording, the state of a resource often cannot be known during command list recording. **AssertResourceState** gives an application developer the ability to impose an assumed state on a resource or subresource at a fixed recording point in a command list.

Often the state of a resource or subresource can either be known due to a previous barrier or inferred-by-use (for example, was used in an earlier call to [CopyBufferRegion](#)) during command list recording. In such cases **AssertResourceState** can produce a debug message if the given state does not match the known or assumed state.

This API is for debug validation only and does not affect the actual runtime or GPU state of the resource.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12DebugCommandList1::GetDebugParameter method (d3d12sdklayers.h)

Article 02/22/2024

Gets optional Command List Debug Layer settings.

Syntax

C++

```
HRESULT GetDebugParameter(
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE Type,
    [out] void* pData,
    UINT DataSize
);
```

Parameters

Type

Type: [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#)

Specifies a [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#) value that determines which debug parameter data to copy to the memory pointed to by *pData*.

[out] *pData*

Type: [void*](#)

Points to the memory that will be filled with a copy of the debug parameter data. The interpretation of this data depends on the [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#) given in the *Type* parameter.

DataSize

Type: [UINT](#)

Size in bytes of the memory buffer pointed to by *pData*.

Return value

Type: [HRESULT](#)

Returns S_OK if successful, otherwise E_INVALIDARG.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList1](#)

[SetDebugParameter](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandList1::SetDebugParameter method (d3d12sdklayers.h)

Article 10/13/2021

Modifies optional Debug Layer settings of a command list.

Syntax

C++

```
HRESULT SetDebugParameter(
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE Type,
    [in] const void* pData,
    UINT DataSize
);
```

Parameters

Type

Type: [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#)

Specifies a [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#) value that indicates which debug parameter data to set.

[in] pData

Type: [const void*](#)

Pointer to debug parameter data to set. The interpretation of this data depends on the [D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE](#) given in the *Type* parameter.

DataSize

Type: [UINT](#)

Specifies the size in bytes of the debug parameter *pData*.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

Certain debug behaviors of D3D12 Debug Layer can be modified by setting debug parameters. These can be used to toggle extra validation or expose experimental debug features.

[ID3D12DebugCommandList1::SetDebugParameter](#) only impacts debug settings for the associated command list. For device-wide debug parameters see the [ID3D12DebugDevice1::SetDebugParameter](#) method.

Resetting a command list restores the debug parameters to the default values. This is because a command list reset is treated as equivalent to creating a new command list.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetDebugParameter](#)

[ID3D12DebugCommandList1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandQueue interface (d3d12sdklayers.h)

Article02/22/2024

Provides methods to monitor and debug a command queue.

Inheritance

The **ID3D12DebugCommandQueue** interface inherits from the [IUnknown](#) interface.

ID3D12DebugCommandQueue also has these types of members:

Methods

The **ID3D12DebugCommandQueue** interface has these methods.

 Expand table

<p>ID3D12DebugCommandQueue::AssertResourceState</p> <p>Checks whether a resource, or subresource, is in a specified state, or not. (ID3D12DebugCommandQueue.AssertResourceState)</p>

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugCommandQueue::AssertResourceState method (d3d12sdklayers.h)

Article 02/22/2024

Checks whether a resource, or subresource, is in a specified state, or not.

Syntax

C++

```
BOOL AssertResourceState(  
    [in] ID3D12Resource *pResource,  
    UINT Subresource,  
    UINT State  
) ;
```

Parameters

[in] pResource

Type: [ID3D12Resource*](#)

Specifies the [ID3D12Resource](#) to check.

Subresource

Type: [UINT](#)

The index of the subresource to check. This can be set to an index, or D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES.

State

Type: [UINT](#)

Specifies the state to check for. This can be one or more D3D12_RESOURCE_STATES flags Or'ed together.

Return value

Type: [BOOL](#)

This method returns true if the resource or subresource is in the specified state, false otherwise.

Remarks

This method is very similar to [ID3D12DebugCommandList::AssertResourceState](#), however there are methods on the command queue that work directly with resources that might need to be monitored (for example [ID3D12CommandQueue::CopyTileMappings](#)).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice interface (d3d12sdklayers.h)

Article02/22/2024

This interface represents a graphics device for debugging.

Inheritance

The `ID3D12DebugDevice` interface inherits from the [IUnknown](#) interface.

`ID3D12DebugDevice` also has these types of members:

Methods

The `ID3D12DebugDevice` interface has these methods.

[] [Expand table](#)

ID3D12DebugDevice::GetFeatureMask
Gets a bit field of flags that indicates which debug features are on or off.
ID3D12DebugDevice::ReportLiveDeviceObjects
Reports information about a device object's lifetime.
ID3D12DebugDevice::SetFeatureMask
Set a bit field of flags that will turn debug features on and off. (<code>ID3D12DebugDevice.SetFeatureMask</code>)

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice::GetFeatureMask method (d3d12sdklayers.h)

Article02/22/2024

Gets a bit field of flags that indicates which debug features are on or off.

Syntax

C++

```
D3D12_DEBUG_FEATURE GetFeatureMask();
```

Return value

Type: [D3D12_DEBUG_FEATURE](#)

Mask of feature-mask flags, as a bitwise OR'ed combination of [D3D12_DEBUG_FEATURE](#) enumeration constants. If a flag is present, then that feature will be set to on, otherwise the feature will be set to off.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice](#)

[ID3D12DebugDevice::SetFeatureMask](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice::ReportLiveDeviceObjects method (d3d12sdklayers.h)

Article 02/22/2024

Reports information about a device object's lifetime.

Syntax

C++

```
HRESULT ReportLiveDeviceObjects(  
    D3D12_RLDO_FLAGS Flags  
);
```

Parameters

Flags

Type: [D3D12_RLDO_FLAGS](#)

A value from the [D3D12_RLDO_FLAGS](#) enumeration. This method uses the value in *Flags* to determine the amount of information to report about a device object's lifetime.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#). [HRESULT](#)

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice::SetFeatureMask method (d3d12sdklayers.h)

Article 02/22/2024

Set a bit field of flags that will turn debug features on and off.

Syntax

C++

```
HRESULT SetFeatureMask(  
    D3D12_DEBUG_FEATURE Mask  
);
```

Parameters

Mask

Type: [D3D12_DEBUG_FEATURE](#)

Feature-mask flags, as a bitwise-OR'd combination of [D3D12_DEBUG_FEATURE](#) enumeration constants. If a flag is present, that feature will be set to on; otherwise, the feature will be set to off.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#). [HRESULT](#)

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetFeatureMask](#)

[ID3D12DebugDevice](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice1 interface (d3d12sdklayers.h)

Article02/22/2024

Specifies device-wide debug layer settings.

Inheritance

The ID3D12DebugDevice1 interface inherits from the [IUnknown](#) interface.

ID3D12DebugDevice1 also has these types of members:

Methods

The ID3D12DebugDevice1 interface has these methods.

[+] Expand table

ID3D12DebugDevice1::GetDebugParameter
Gets optional device-wide Debug Layer settings.
ID3D12DebugDevice1::ReportLiveDeviceObjects
Specifies the amount of information to report on a device object's lifetime.
ID3D12DebugDevice1::SetDebugParameter
Modifies the D3D12 optional device-wide Debug Layer settings.

Remarks

This interface is currently in Preview mode.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice1::GetDebugParameter method (d3d12sdklayers.h)

Article 10/13/2021

Gets optional device-wide Debug Layer settings.

Syntax

C++

```
HRESULT GetDebugParameter(  
    D3D12_DEBUG_DEVICE_PARAMETER_TYPE Type,  
    [out] void* pData,  
    UINT DataSize  
>;
```

Parameters

Type

Type: [D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#)

Specifies a [D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#) value that indicates which debug parameter data to set.

[out] pData

Type: [void*](#)

Points to the memory that will be filled with a copy of the debug parameter data. The interpretation of this data depends on the [D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#) given in the *Type* parameter.

DataSize

Type: [UINT](#)

Size in bytes of the memory buffer pointed to by *pData*.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice1](#)

[SetDebugParameter](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice1::ReportLiveDeviceObjects method (d3d12sdklayers.h)

Article 02/22/2024

Specifies the amount of information to report on a device object's lifetime.

Syntax

C++

```
HRESULT ReportLiveDeviceObjects(  
    D3D12_RLDO_FLAGS Flags  
);
```

Parameters

Flags

Type: [D3D12_RLDO_FLAGS](#)

A value from the [D3D12_RLDO_FLAGS](#) enumeration. This method uses the value in *Flags* to determine the amount of information to report about a device object's lifetime.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12DebugDevice1::SetDebugParameter method (d3d12sdklayers.h)

Article 02/22/2024

Modifies the D3D12 optional device-wide Debug Layer settings.

Syntax

C++

```
HRESULT SetDebugParameter(
    D3D12_DEBUG_DEVICE_PARAMETER_TYPE Type,
    [in] const void*                 pData,
    UINT                           DataSize
);
```

Parameters

Type

Type: [D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#)

Specifies a [D3D12_DEBUG_DEVICE_PARAMETER_TYPE](#) value that indicates which debug parameter data to get.

[in] pData

Type: [const void*](#)

Debug parameter data to set.

DataSize

Type: [UINT](#)

Size in bytes of the data pointed to by *pData*.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetDebugParameter](#)

[ID3D12DebugDevice1](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue interface (d3d12sdklayers.h)

Article 07/27/2022

An information-queue interface stores, retrieves, and filters debug messages. The queue consists of a message queue, an optional storage filter stack, and a optional retrieval filter stack.

Inheritance

The **ID3D12InfoQueue** interface inherits from the [IUnknown](#) interface. **ID3D12InfoQueue** also has these types of members:

Methods

The **ID3D12InfoQueue** interface has these methods.

[+] Expand table

ID3D12InfoQueue::AddApplicationMessage
Adds a user-defined message to the message queue and sends that message to debug output.
ID3D12InfoQueue::AddMessage
Adds a debug message to the message queue and sends that message to debug output.
ID3D12InfoQueue::AddRetrievalFilterEntries
Add storage filters to the top of the retrieval-filter stack. (ID3D12InfoQueue.AddRetrievalFilterEntries)
ID3D12InfoQueue::AddStorageFilterEntries
Add storage filters to the top of the storage-filter stack. (ID3D12InfoQueue.AddStorageFilterEntries)
ID3D12InfoQueue::ClearRetrievalFilter
Remove a retrieval filter from the top of the retrieval-filter stack. (ID3D12InfoQueue.ClearRetrievalFilter)

[ID3D12InfoQueue::ClearStorageFilter](#)

Remove a storage filter from the top of the storage-filter stack.
(ID3D12InfoQueue.ClearStorageFilter)

[ID3D12InfoQueue::ClearStoredMessages](#)

Clear all messages from the message queue. (ID3D12InfoQueue.ClearStoredMessages)

[ID3D12InfoQueue::GetBreakOnCategory](#)

Get a message category to break on when a message with that category passes through the storage filter. (ID3D12InfoQueue.GetBreakOnCategory)

[ID3D12InfoQueue::GetBreakOnID](#)

Get a message identifier to break on when a message with that identifier passes through the storage filter. (ID3D12InfoQueue.GetBreakOnID)

[ID3D12InfoQueue::GetBreakOnSeverity](#)

Get a message severity level to break on when a message with that severity level passes through the storage filter. (ID3D12InfoQueue.GetBreakOnSeverity)

[ID3D12InfoQueue::GetMessage](#)

Get a message from the message queue. (ID3D12InfoQueue.GetMessage)

[ID3D12InfoQueue::GetMessageCountLimit](#)

Get the maximum number of messages that can be added to the message queue.
(ID3D12InfoQueue.GetMessageCountLimit)

[ID3D12InfoQueue::GetMuteDebugOutput](#)

Get a boolean that determines if debug output is on or off.

[ID3D12InfoQueue::GetNumMessagesAllowedByStorageFilter](#)

Get the number of messages that were allowed to pass through a storage filter.
(ID3D12InfoQueue.GetNumMessagesAllowedByStorageFilter)

[ID3D12InfoQueue::GetNumMessagesDeniedByStorageFilter](#)

Get the number of messages that were denied passage through a storage filter.
(ID3D12InfoQueue.GetNumMessagesDeniedByStorageFilter)

[ID3D12InfoQueue::GetNumMessagesDiscardedByMessageCountLimit](#)

Get the number of messages that were discarded due to the message count limit.
(ID3D12InfoQueue.GetNumMessagesDiscardedByMessageCountLimit)

[ID3D12InfoQueue::GetNumStoredMessages](#)

Get the number of messages currently stored in the message queue.
(ID3D12InfoQueue.GetNumStoredMessages)

[ID3D12InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter](#)

Get the number of messages that are able to pass through a retrieval filter.
(ID3D12InfoQueue.GetNumStoredMessagesAllowedByRetrievalFilter)

[ID3D12InfoQueue::GetRetrievalFilter](#)

Get the retrieval filter at the top of the retrieval-filter stack. (ID3D12InfoQueue.GetRetrievalFilter)

[ID3D12InfoQueue::GetRetrievalFilterStackSize](#)

Get the size of the retrieval-filter stack in bytes. (ID3D12InfoQueue.GetRetrievalFilterStackSize)

[ID3D12InfoQueue::GetStorageFilter](#)

Get the storage filter at the top of the storage-filter stack. (ID3D12InfoQueue.GetStorageFilter)

[ID3D12InfoQueue::GetStorageFilterStackSize](#)

Get the size of the storage-filter stack in bytes. (ID3D12InfoQueue.GetStorageFilterStackSize)

[ID3D12InfoQueue::PopRetrievalFilter](#)

Pop a retrieval filter from the top of the retrieval-filter stack. (ID3D12InfoQueue.PopRetrievalFilter)

[ID3D12InfoQueue::PopStorageFilter](#)

Pop a storage filter from the top of the storage-filter stack. (ID3D12InfoQueue.PopStorageFilter)

[ID3D12InfoQueue::PushCopyOfRetrievalFilter](#)

Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack. (ID3D12InfoQueue.PushCopyOfRetrievalFilter)

[ID3D12InfoQueue::PushCopyOfStorageFilter](#)

Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack. (ID3D12InfoQueue.PushCopyOfStorageFilter)

[ID3D12InfoQueue::PushEmptyRetrievalFilter](#)

	<p>Push an empty retrieval filter onto the retrieval-filter stack. (ID3D12InfoQueue.PushEmptyRetrievalFilter)</p>
	<p>ID3D12InfoQueue::PushEmptyStorageFilter</p>
	<p>Push an empty storage filter onto the storage-filter stack. (ID3D12InfoQueue.PushEmptyStorageFilter)</p>
	<p>ID3D12InfoQueue::PushRetrievalFilter</p>
	<p>Push a retrieval filter onto the retrieval-filter stack. (ID3D12InfoQueue.PushRetrievalFilter)</p>
	<p>ID3D12InfoQueue::PushStorageFilter</p>
	<p>Push a storage filter onto the storage-filter stack. (ID3D12InfoQueue.PushStorageFilter)</p>
	<p>ID3D12InfoQueue::SetBreakOnCategory</p>
	<p>Set a message category to break on when a message with that category passes through the storage filter. (ID3D12InfoQueue.SetBreakOnCategory)</p>
	<p>ID3D12InfoQueue::SetBreakOnID</p>
	<p>Set a message identifier to break on when a message with that identifier passes through the storage filter. (ID3D12InfoQueue.SetBreakOnID)</p>
	<p>ID3D12InfoQueue::SetBreakOnSeverity</p>
	<p>Set a message severity level to break on when a message with that severity level passes through the storage filter. (ID3D12InfoQueue.SetBreakOnSeverity)</p>
	<p>ID3D12InfoQueue::SetMessageCountLimit</p>
	<p>Set the maximum number of messages that can be added to the message queue. (ID3D12InfoQueue.SetMessageCountLimit)</p>
	<p>ID3D12InfoQueue::SetMuteDebugOutput</p>
	<p>Set a boolean that turns the debug output on or off. (ID3D12InfoQueue.SetMuteDebugOutput)</p>

Remarks

This interface is obtained by querying it from the [ID3D12Device](#) using [IUnknown::QueryInterface](#). The [ID3D12Debug](#) layer must be enabled through [ID3D12Debug::EnableDebugLayer](#) for that operation to succeed.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::AddApplicationMessage method (d3d12sdklayers.h)

Article 02/22/2024

Adds a user-defined message to the message queue and sends that message to debug output.

Syntax

C++

```
HRESULT AddApplicationMessage(
    [in] D3D12_MESSAGE_SEVERITY Severity,
    [in] LPCSTR             pDescription
);
```

Parameters

[in] Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

Severity of a message.

[in] pDescription

Type: [LPCSTR](#)

Specifies the message string.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::AddMessage method (d3d12sdklayers.h)

Article 02/22/2024

Adds a debug message to the message queue and sends that message to debug output.

Syntax

C++

```
HRESULT AddMessage(  
    [in] D3D12_MESSAGE_CATEGORY Category,  
    [in] D3D12_MESSAGE_SEVERITY Severity,  
    [in] D3D12_MESSAGE_ID       ID,  
    [in] LPCSTR                 pDescription  
) ;
```

Parameters

[in] Category

Type: [D3D12_MESSAGE_CATEGORY](#)

Category of a message.

[in] Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

Severity of a message.

[in] ID

Type: [D3D12_MESSAGE_ID](#)

Unique identifier of a message.

[in] pDescription

Type: [LPCSTR](#)

User-defined message.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method is used by the runtime's internal mechanisms to add debug messages to the message queue and send them to debug output. For applications to add their own custom messages to the message queue and send them to debug output, call [ID3D12InfoQueue::AddApplicationMessage](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::AddRetrievalFilterEntries method (d3d12sdklayers.h)

Article 02/22/2024

Add storage filters to the top of the retrieval-filter stack.

Syntax

C++

```
HRESULT AddRetrievalFilterEntries(
    [in] D3D12_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Array of retrieval filters.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

The following code example shows how to use this method:

syntax

```
D3D12_MESSAGE_CATEGORY cats[] = { ..., ..., ... };
D3D12_MESSAGE_SEVERITY sevs[] = { ..., ..., ... };
D3D12_MESSAGE_ID ids[] = { ..., ..., ... };

D3D12_INFO_QUEUE_FILTER filter;
memset( &filter, 0, sizeof(filter) );
```

```
// To set the type of messages to allow,  
// set filter.AllowList as follows:  
filter.AllowList.NumCategories = _countof(cats);  
filter.AllowList.pCategoryList = cats;  
filter.AllowList.NumSeverities = _countof(sevs);  
filter.AllowList.pSeverityList = sevs;  
filter.AllowList.NumIDs = _countof(ids);  
filter.AllowList.pIDList = ids;  
  
// To set the type of messages to deny, set filter.DenyList  
// similarly to the preceding filter.AllowList.  
  
// The following single call sets all of the preceding information.  
hr = infoQueue->AddRetrievalFilterEntries( &filter );
```

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::AddStorageFilterEntries method (d3d12sdklayers.h)

Article 02/22/2024

Add storage filters to the top of the storage-filter stack.

Syntax

C++

```
HRESULT AddStorageFilterEntries(
    [in] D3D12_INFO_QUEUE_FILTER *pFilter
);
```

Parameters

[in] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Array of storage filters.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::ClearRetrievalFilter method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
void ClearRetrievalFilter();
```

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::ClearStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
void ClearStorageFilter();
```

Return value

None

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::ClearStoredMessages method (d3d12sdklayers.h)

Article 02/22/2024

Clear all messages from the message queue.

Syntax

C++

```
void ClearStoredMessages();
```

Return value

None

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetBreakOnCategory method (d3d12sdklayers.h)

Article 02/22/2024

Get a message category to break on when a message with that category passes through the storage filter.

Syntax

C++

```
BOOL GetBreakOnCategory(  
    [in] D3D12_MESSAGE_CATEGORY Category  
);
```

Parameters

[in] Category

Type: [D3D12_MESSAGE_CATEGORY](#)

Message category to break on.

Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetBreakOnID method (d3d12sdklayers.h)

Article 02/22/2024

Get a message identifier to break on when a message with that identifier passes through the storage filter.

Syntax

C++

```
BOOL GetBreakOnID(  
    [in] D3D12_MESSAGE_ID ID  
);
```

Parameters

[in] ID

Type: [D3D12_MESSAGE_ID](#)

Message identifier to break on.

Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetBreakOnSeverity method (d3d12sdklayers.h)

Article 02/22/2024

Get a message severity level to break on when a message with that severity level passes through the storage filter.

Syntax

C++

```
BOOL GetBreakOnSeverity(  
    [in] D3D12_MESSAGE_SEVERITY Severity  
);
```

Parameters

[in] Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

Message severity level to break on.

Return value

Type: [BOOL](#)

Whether this breaking condition is turned on or off (true for on, false for off).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetMessage method (d3d12sdklayers.h)

Article 02/22/2024

Get a message from the message queue.

Syntax

C++

```
HRESULT GetMessage(
    [in]          UINT64      MessageIndex,
    [out, optional] D3D12_MESSAGE *pMessage,
    [in, out]       SIZE_T      *pMessageByteLength
);
```

Parameters

[in] MessageIndex

Type: **UINT64**

Index into 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 (which can be obtained with

[GetNumStoredMessagesAllowedByRetrievalFilter](#)). 0 is the message at the front of the message queue.

[out, optional] pMessage

Type: **D3D12_MESSAGE***

Returned message.

[in, out] pMessageByteLength

Type: **SIZE_T***

Size of *pMessage* in bytes.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method does not remove any messages from the message queue.

This method gets messages from the message queue after an optional retrieval filter has been applied.

Applications should 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:

syntax

```
// Get the size of the message
SIZE_T messageLength = 0;
HRESULT hr = pInfoQueue->GetMessage(0, NULL, &messageLength);

// Allocate space and get the message
D3D12_MESSAGE * pMessage = (D3D12_MESSAGE*)malloc(messageLength);
hr = pInfoQueue->GetMessage(0, pMessage, &messageLength);
```

Requirements

[\[\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12InfoQueue::GetMessageCountLimit method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
UINT64 GetMessageCountLimit();
```

Return value

Type: **UINT64**

Maximum number of messages that can be added to the queue. -1 means no limit.

When the number of messages in the message queue has reached the maximum limit, new messages coming in will push old messages out.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12InfoQueue::GetMuteDebugOutput method (d3d12sdklayers.h)

Article02/22/2024

Get a boolean that determines if debug output is on or off.

Syntax

C++

```
BOOL GetMuteDebugOutput();
```

Return value

Type: **BOOL**

Whether the debug output is on or off (true for on, false for off).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

thumb up Yes

thumb down No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetNumMessagesAllowedByStorageFilter method (d3d12sdklayers.h)

Article02/22/2024

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

Syntax

C++

```
UINT64 GetNumMessagesAllowedByStorageFilter();
```

Return value

Type: **UINT64**

Number of messages allowed by a storage filter.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12InfoQueue::GetNumMessagesDeniedByStorageFilter method (d3d12sdklayers.h)

Article02/22/2024

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

Syntax

C++

```
UINT64 GetNumMessagesDeniedByStorageFilter();
```

Return value

Type: **UINT64**

Number of messages denied by a storage filter.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12InfoQueue::GetNumMessagesDiscardedByMessageCountLimit method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
UINT64 GetNumMessagesDiscardedByMessageCountLimit();
```

Return value

Type: **UINT64**

Number of messages discarded.

Remarks

Get and set the message count limit with [GetMessageCountLimit](#) and [SetMessageCountLimit](#), respectively.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetNumStoredMessages method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
UINT64 GetNumStoredMessages();
```

Return value

Type: **UINT64**

Number of messages currently stored in the message queue.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter method (d3d12sdklayers.h)

Article02/22/2024

Get the number of messages that are able to pass through a retrieval filter.

Syntax

C++

```
UINT64 GetNumStoredMessagesAllowedByRetrievalFilter();
```

Return value

Type: **UINT64**

Number of messages allowed by a retrieval filter.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12InfoQueue::GetRetrievalFilter method (d3d12sdklayers.h)

Article 02/22/2024

Get the retrieval filter at the top of the retrieval-filter stack.

Syntax

C++

```
HRESULT GetRetrievalFilter(
    [out, optional] D3D12_INFO_QUEUE_FILTER *pFilter,
    [in, out]          SIZE_T             *pFilterByteLength
);
```

Parameters

[out, optional] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Retrieval filter at the top of the retrieval-filter stack.

[in, out] pFilterByteLength

Type: [SIZE_T*](#)

Size of the retrieval filter in bytes. If *pFilter* is NULL, the size of the retrieval filter will be output to this parameter.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetRetrievalFilterStackSize method (d3d12sdklayers.h)

Article02/22/2024

Get the size of the retrieval-filter stack in bytes.

Syntax

C++

```
UINT GetRetrievalFilterStackSize();
```

Return value

Type: **UINT**

Size of the retrieval-filter stack in bytes.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

Get the storage filter at the top of the storage-filter stack.

Syntax

C++

```
HRESULT GetStorageFilter(
    [out, optional] D3D12_INFO_QUEUE_FILTER *pFilter,
    [in, out]          SIZE_T             *pFilterByteLength
);
```

Parameters

[out, optional] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Storage filter at the top of the storage-filter stack.

[in, out] pFilterByteLength

Type: [SIZE_T*](#)

Size of the storage filter in bytes. If *pFilter* is NULL, the size of the storage filter will be output to this parameter.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::GetStorageFilterStackSize method (d3d12sdklayers.h)

Article02/22/2024

Get the size of the storage-filter stack in bytes.

Syntax

C++

```
UINT GetStorageFilterStackSize();
```

Return value

Type: **UINT**

Size of the storage-filter stack in bytes.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PopRetrievalFilter method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
void PopRetrievalFilter();
```

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PopStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
void PopStorageFilter();
```

Return value

None

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PushCopyOfRetrievalFilter method (d3d12sdklayers.h)

Article02/22/2024

Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushCopyOfRetrievalFilter();
```

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12InfoQueue::PushCopyOfStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack.

Syntax

C++

```
HRESULT PushCopyOfStorageFilter();
```

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12InfoQueue::PushEmptyRetrievalFilter method (d3d12sdklayers.h)

Article 02/22/2024

Push an empty retrieval filter onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushEmptyRetrievalFilter();
```

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

An empty retrieval filter allows all messages to pass through.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PushEmptyStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

Push an empty storage filter onto the storage-filter stack.

Syntax

C++

```
HRESULT PushEmptyStorageFilter();
```

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Remarks

An empty storage filter allows all messages to pass through.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PushRetrievalFilter method (d3d12sdklayers.h)

Article 02/22/2024

Push a retrieval filter onto the retrieval-filter stack.

Syntax

C++

```
HRESULT PushRetrievalFilter(  
    [in] D3D12_INFO_QUEUE_FILTER *pFilter  
);
```

Parameters

[in] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Pointer to a retrieval filter.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::PushStorageFilter method (d3d12sdklayers.h)

Article 02/22/2024

Push a storage filter onto the storage-filter stack.

Syntax

C++

```
HRESULT PushStorageFilter(  
    [in] D3D12_INFO_QUEUE_FILTER *pFilter  
);
```

Parameters

[in] pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Pointer to a storage filter.

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::SetBreakOnCategory method (d3d12sdklayers.h)

Article 02/22/2024

Set a message category to break on when a message with that category passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnCategory(
    [in] D3D12_MESSAGE_CATEGORY Category,
    [in] BOOL                 bEnable
);
```

Parameters

[in] Category

Type: [D3D12_MESSAGE_CATEGORY](#)

Message category to break on.

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::SetBreakOnID method (d3d12sdklayers.h)

Article 02/22/2024

Set a message identifier to break on when a message with that identifier passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnID(  
    [in] D3D12_MESSAGE_ID ID,  
    [in] BOOL             bEnable  
);
```

Parameters

[in] ID

Type: [D3D12_MESSAGE_ID](#)

Message identifier to break on.

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::SetBreakOnSeverity method (d3d12sdklayers.h)

Article 02/22/2024

Set a message severity level to break on when a message with that severity level passes through the storage filter.

Syntax

C++

```
HRESULT SetBreakOnSeverity(  
    [in] D3D12_MESSAGE_SEVERITY Severity,  
    [in] BOOL bEnable  
);
```

Parameters

[in] Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

A message severity level to break on.

[in] bEnable

Type: [BOOL](#)

Turns this breaking condition on or off (true for on, false for off).

Return value

Type: [HRESULT](#)

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::SetMessageCountLimit method (d3d12sdklayers.h)

Article 02/22/2024

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

Syntax

C++

```
HRESULT SetMessageCountLimit(  
    [in] UINT64 MessageCountLimit  
);
```

Parameters

[in] `MessageCountLimit`

Type: `UINT64`

Maximum number of messages that can be added to the message queue. -1 means no limit.

When the number of messages in the message queue has reached the maximum limit, new messages coming in will push old messages out.

Return value

Type: `HRESULT`

This method returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12InfoQueue::SetMuteDebugOutput method (d3d12sdklayers.h)

Article 02/22/2024

Set a boolean that turns the debug output on or off.

Syntax

C++

```
void SetMuteDebugOutput(  
    [in] BOOL bMute  
);
```

Parameters

[in] bMute

Type: **BOOL**

Disable/Enable the debug output (true to disable or mute the output, false to enable the output).

Return value

None

Remarks

This will stop messages that pass the storage filter from being printed out in the debug output, however those messages will still be added to the message queue.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12SharingContract interface (d3d12sdklayers.h)

Article 02/22/2024

Part of a contract between D3D11On12 diagnostic layers and graphics diagnostics tools. This interface facilitates diagnostics tools to capture information at a lower level than the DXGI swapchain.

You may want to use this interface to enable diagnostic tools to capture usage patterns that don't use DXGI swap chains for presentation. If so, you can access this interface via **QueryInterface** from a D3D12 command queue. Note that this interface is not supported when there are no diagnostic tools present, so your application mustn't rely on it existing.

Inheritance

The **ID3D12SharingContract** interface inherits from the [IUnknown](#) interface.

ID3D12SharingContract also has these types of members:

Methods

The **ID3D12SharingContract** interface has these methods.

[] [Expand table](#)

ID3D12SharingContract::Present Shares a resource (or subresource) between the D3D layers and diagnostics tools.
ID3D12SharingContract::SharedFenceSignal Signals a shared fence between the D3D layers and diagnostics tools.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h (include D3D12.h)

See also

[Core interfaces](#), [IUnknown](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12SharingContract::Present method (d3d12sdklayers.h)

Article 02/22/2024

Notifies diagnostic tools about an end-of-frame operation without the use of a swap chain. Calling this API enables usage of tools like PIX with applications that either don't render to a window, or that do so in non-traditional ways.

Syntax

C++

```
void Present(
    [in] ID3D12Resource *pResource,
    UINT             Subresource,
    HWND             window
);
```

Parameters

[in] pResource

Type: [ID3D12Resource*](#)

A pointer to the resource that contains the final frame contents. This resource is treated as the *back buffer* of the **Present**.

Subresource

Type: [UINT](#)

An unsigned 32bit subresource id.

window

If provided, indicates which window the tools should use for displaying additional diagnostic information.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12sdklayers.h (include D3D12.h)

See also

[ID3D12SharingContract](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12SharingContract::SharedFenceSignal method (d3d12sdklayers.h)

Article 02/22/2024

Signals a shared fence between the D3D layers and diagnostics tools.

Syntax

C++

```
void SharedFenceSignal(  
    [in] ID3D12Fence *pFence,  
    UINT64      FenceValue  
) ;
```

Parameters

[in] pFence

Type: [ID3D12Fence*](#)

A pointer to the shared fence to signal.

FenceValue

Type: [UINT64](#)

An unsigned 64bit value to signal the shared fence with.

Return value

None

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12sdklayers.h (include D3D12.h)

See also

[ID3D12SharingContract](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

d3d12shader.h header

Article 01/24/2023

This header is used by Direct3D 12 Graphics. For more information, see:

- [Direct3D 12 Graphics](#)

d3d12shader.h contains the following programming interfaces:

Interfaces

[ID3D12FunctionParameterReflection](#)

A function-parameter-reflection interface accesses function-parameter info.
([ID3D12FunctionParameterReflection](#))

[ID3D12FunctionReflection](#)

A function-reflection interface accesses function info. ([ID3D12FunctionReflection](#))

[ID3D12LibraryReflection](#)

A library-reflection interface accesses library info. ([ID3D12LibraryReflection](#))

[ID3D12ShaderReflection](#)

A shader-reflection interface accesses shader information. ([ID3D12ShaderReflection](#))

[ID3D12ShaderReflectionConstantBuffer](#)

This shader-reflection interface provides access to a constant buffer.
([ID3D12ShaderReflectionConstantBuffer](#))

[ID3D12ShaderReflectionType](#)

This shader-reflection interface provides access to variable type. ([ID3D12ShaderReflectionType](#))

[ID3D12ShaderReflectionVariable](#)

This shader-reflection interface provides access to a variable. ([ID3D12ShaderReflectionVariable](#))

Structures

[D3D12_FUNCTION_DESC](#)

Describes a function. (D3D12_FUNCTION_DESC)

[D3D12_LIBRARY_DESC](#)

Describes a library. (D3D12_LIBRARY_DESC)

[D3D12_PARAMETER_DESC](#)

Describes a function parameter. (D3D12_PARAMETER_DESC)

[D3D12_SHADER_BUFFER_DESC](#)

Describes a shader constant-buffer. (D3D12_SHADER_BUFFER_DESC)

[D3D12_SHADER_DESC](#)

Describes a shader. (D3D12_SHADER_DESC)

[D3D12_SHADER_INPUT_BIND_DESC](#)

Describes how a shader resource is bound to a shader input. (D3D12_SHADER_INPUT_BIND_DESC)

[D3D12_SHADER_TYPE_DESC](#)

Describes a shader-variable type. (D3D12_SHADER_TYPE_DESC)

[D3D12_SHADER_VARIABLE_DESC](#)

Describes a shader variable. (D3D12_SHADER_VARIABLE_DESC)

[D3D12_SIGNATURE_PARAMETER_DESC](#)

Describes a shader signature. (D3D12_SIGNATURE_PARAMETER_DESC)

Enumerations

[D3D12_SHADER_VERSION_TYPE](#)

Enumerates the types of shaders that Direct3D recognizes. Used to encode the Version member of the D3D12_SHADER_DESC structure.

Feedback

Was this page helpful?

 Yes

 No

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

D3D12_FUNCTION_DESC structure (d3d12shader.h)

Article 07/27/2022

Describes a function.

Syntax

C++

```
typedef struct _D3D12_FUNCTION_DESC {
    UINT             Version;
    LPCSTR           Creator;
    UINT             Flags;
    UINT             ConstantBuffers;
    UINT             BoundResources;
    UINT             InstructionCount;
    UINT             TempRegisterCount;
    UINT             TempArrayCount;
    UINT             DefCount;
    UINT             DclCount;
    UINT             TextureNormalInstructions;
    UINT             TextureLoadInstructions;
    UINT             TextureCompInstructions;
    UINT             TextureBiasInstructions;
    UINT             TextureGradientInstructions;
    UINT             FloatInstructionCount;
    UINT             IntInstructionCount;
    UINT             UintInstructionCount;
    UINT             StaticFlowControlCount;
    UINT             DynamicFlowControlCount;
    UINT             MacroInstructionCount;
    UINT             ArrayInstructionCount;
    UINT             MovInstructionCount;
    UINT             MovcInstructionCount;
    UINT             ConversionInstructionCount;
    UINT             BitwiseInstructionCount;
    D3D_FEATURE_LEVEL MinFeatureLevel;
    UINT64           RequiredFeatureFlags;
    LPCSTR           Name;
    INT              FunctionParameterCount;
    BOOL             HasReturn;
    BOOL             Has10Level9VertexShader;
    BOOL             Has10Level9PixelShader;
} D3D12_FUNCTION_DESC;
```

Members

Version

The shader version. See also [D3D12_SHADER_VERSION_TYPE](#).

Creator

The name of the originator of the function.

Flags

A combination of [D3DCOMPILE Constants](#) that are combined by using a bitwise OR operation. The resulting value specifies shader compilation and parsing.

ConstantBuffers

The number of constant buffers for the function.

BoundResources

The number of bound resources for the function.

InstructionCount

The number of emitted instructions for the function.

TempRegisterCount

The number of temporary registers used by the function.

TempArrayCount

The number of temporary arrays used by the function.

DefCount

The number of constant defines for the function.

DclCount

The number of declarations (input + output) for the function.

TextureNormalInstructions

The number of non-categorized texture instructions for the function.

TextureLoadInstructions

The number of texture load instructions for the function.

`TextureCompInstructions`

The number of texture comparison instructions for the function.

`TextureBiasInstructions`

The number of texture bias instructions for the function.

`TextureGradientInstructions`

The number of texture gradient instructions for the function.

`FloatInstructionCount`

The number of floating point arithmetic instructions used by the function.

`IntInstructionCount`

The number of signed integer arithmetic instructions used by the function.

`UintInstructionCount`

The number of unsigned integer arithmetic instructions used by the function.

`StaticFlowControlCount`

The number of static flow control instructions used by the function.

`DynamicFlowControlCount`

The number of dynamic flow control instructions used by the function.

`MacroInstructionCount`

The number of macro instructions used by the function.

`ArrayInstructionCount`

The number of array instructions used by the function.

`MovInstructionCount`

The number of mov instructions used by the function.

`MovcInstructionCount`

The number of movc instructions used by the function.

`ConversionInstructionCount`

The number of type conversion instructions used by the function.

`BitwiseInstructionCount`

The number of bitwise arithmetic instructions used by the function.

`MinFeatureLevel`

A [D3D_FEATURE_LEVEL](#)-typed value that specifies the minimum Direct3D feature level target of the function byte code.

`RequiredFeatureFlags`

A value that contains a combination of one or more shader requirements flags; each flag specifies a requirement of the shader. A default value of 0 means there are no requirements. For a list of values, see [ID3D12ShaderReflection::GetRequiresFlags](#).

`Name`

The name of the function.

`FunctionParameterCount`

The number of logical parameters in the function signature, not including the return value.

`HasReturn`

Indicates whether the function returns a value. **TRUE** indicates it returns a value; otherwise, **FALSE** (it is a subroutine).

`Has10Level9VertexShader`

Indicates whether there is a Direct3D 10Level9 vertex shader blob. **TRUE** indicates there is a 10Level9 vertex shader blob; otherwise, **FALSE**.

`Has10Level9PixelShader`

Indicates whether there is a Direct3D 10Level9 pixel shader blob. **TRUE** indicates there is a 10Level9 pixel shader blob; otherwise, **FALSE**.

Remarks

This structure is returned by [ID3D12FunctionReflection::GetDesc](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12shader.h

See also

[ID3D12FunctionReflection::GetDesc](#)

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_LIBRARY_DESC structure (d3d12shader.h)

Article 02/22/2024

Describes a library.

Syntax

C++

```
typedef struct _D3D12_LIBRARY_DESC {
    LPCSTR Creator;
    UINT    Flags;
    UINT    FunctionCount;
} D3D12_LIBRARY_DESC;
```

Members

Creator

The name of the originator of the library.

Flags

A combination of [D3DCOMPILE Constants](#) that are combined by using a bitwise OR operation. The resulting value specifies how the compiler compiles.

FunctionCount

The number of functions exported from the library.

Remarks

This structure is returned by [ID3D12LibraryReflection::GetDesc](#).

Requirements

[] [Expand table](#)

Requirement	Value
Header	d3d12shader.h

See also

[ID3D12LibraryReflection::GetDesc](#)

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

D3D12_PARAMETER_DESC structure (d3d12shader.h)

Article 02/22/2024

Describes a function parameter.

Syntax

C++

```
typedef struct _D3D12_PARAMETER_DESC {
    LPCSTR             Name;
    LPCSTR             SemanticName;
    D3D_SHADER_VARIABLE_TYPE Type;
    D3D_SHADER_VARIABLE_CLASS Class;
    UINT               Rows;
    UINT               Columns;
    D3D_INTERPOLATION_MODE InterpolationMode;
    D3D_PARAMETER_FLAGS Flags;
    UINT               FirstInRegister;
    UINT               FirstInComponent;
    UINT               FirstOutRegister;
    UINT               FirstOutComponent;
} D3D12_PARAMETER_DESC;
```

Members

Name

The name of the function parameter.

SemanticName

The HLSL [semantic](#) that is associated with this function parameter. This name includes the index, for example, SV_Target[n].

Type

A [D3D_SHADER_VARIABLE_TYPE](#)-typed value that identifies the variable type for the parameter.

Class

A [D3D_SHADER_VARIABLE_CLASS](#)-typed value that identifies the variable class for the parameter as one of scalar, vector, matrix, object, and so on.

Rows

The number of rows for a matrix parameter.

Columns

The number of columns for a matrix parameter.

InterpolationMode

A [D3D_INTERPOLATION_MODE](#)-typed value that identifies the interpolation mode for the parameter.

Flags

A combination of [D3D_PARAMETER_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies semantic flags for the parameter.

FirstInRegister

The first input register for this parameter.

FirstInComponent

The first input register component for this parameter.

FirstOutRegister

The first output register for this parameter.

FirstOutComponent

The first output register component for this parameter.

Remarks

Get a function-parameter description by calling [ID3D12FunctionParameterReflection::GetDesc](#).

Requirements

[] Expand table

Requirement	Value
Header	d3d12shader.h

See also

[ID3D12FunctionParameterReflection::GetDesc](#)

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_BUFFER_DESC structure (d3d12shader.h)

Article 02/22/2024

Describes a shader constant-buffer.

Syntax

C++

```
typedef struct _D3D12_SHADER_BUFFER_DESC {
    LPCSTR           Name;
    D3D_CBUFFER_TYPE Type;
    UINT             Variables;
    UINT             Size;
    UINT             uFlags;
} D3D12_SHADER_BUFFER_DESC;
```

Members

Name

The name of the buffer.

Type

A [D3D_CBUFFER_TYPE](#)-typed value that indicates the intended use of the constant data.

Variables

The number of unique variables.

Size

The size of the buffer, in bytes.

uFlags

A combination of [D3D_SHADER_CBUFFER_FLAGS](#)-typed values that are combined by using a bitwise OR operation. The resulting value specifies properties for the shader constant-buffer.

Remarks

Constants are supplied to shaders in a shader-constant buffer. Get the description of a shader-constant-buffer by calling [ID3D12ShaderReflectionConstantBuffer::GetDesc](#).

Requirements

[+] Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_DESC structure (d3d12shader.h)

Article07/27/2022

Describes a shader.

Syntax

C++

```
typedef struct _D3D12_SHADER_DESC {
    UINT Version;
    LPCSTR Creator;
    Flags;
    ConstantBuffers;
    BoundResources;
    InputParameters;
    OutputParameters;
    InstructionCount;
    TempRegisterCount;
    TempArrayCount;
    DefCount;
    DclCount;
    TextureNormalInstructions;
    TextureLoadInstructions;
    TextureCompInstructions;
    TextureBiasInstructions;
    TextureGradientInstructions;
    FloatInstructionCount;
    IntInstructionCount;
    UintInstructionCount;
    StaticFlowControlCount;
    DynamicFlowControlCount;
    MacroInstructionCount;
    ArrayInstructionCount;
    CutInstructionCount;
    EmitInstructionCount;
    GSOutputTopology;
    GSMaxOutputVertexCount;
    InputPrimitive;
    PatchConstantParameters;
    cGSInstanceCount;
    cControlPoints;
    HSOutputPrimitive;
    HSPartitioning;
    TessellatorDomain;
    cBarrierInstructions;
    cInterlockedInstructions;
}
```

```
    UINT                      cTextureStoreInstructions;
} D3D12_SHADER_DESC;
```

Members

Version

The Shader version, as an encoded `UINT` that corresponds to a shader model, such as "ps_5_0". **Version** describes the program type, a major version number, and a minor version number. The program type is a `D3D12_SHADER_VERSION_TYPE` enumeration constant. **Version** is decoded in the following way:

- Program type = (`Version` & 0xFFFF0000) >> 16
- Major version = (`Version` & 0x000000F0) >> 4
- Minor version = (`Version` & 0x0000000F)

Creator

The name of the originator of the shader.

Flags

Shader compilation/parse flags.

ConstantBuffers

The number of shader-constant buffers.

BoundResources

The number of resource (textures and buffers) bound to a shader.

InputParameters

The number of parameters in the input signature.

OutputParameters

The number of parameters in the output signature.

InstructionCount

The number of intermediate-language instructions in the compiled shader.

TempRegisterCount

The number of temporary registers in the compiled shader.

`TempArrayCount`

Number of temporary arrays used.

`DefCount`

Number of constant defines.

`DclCount`

Number of declarations (input + output).

`TextureNormalInstructions`

Number of non-categorized texture instructions.

`TextureLoadInstructions`

Number of texture load instructions

`TextureCompInstructions`

Number of texture comparison instructions

`TextureBiasInstructions`

Number of texture bias instructions

`TextureGradientInstructions`

Number of texture gradient instructions.

`FloatInstructionCount`

Number of floating point arithmetic instructions used.

`IntInstructionCount`

Number of signed integer arithmetic instructions used.

`UintInstructionCount`

Number of unsigned integer arithmetic instructions used.

`StaticFlowControlCount`

Number of static flow control instructions used.

`DynamicFlowControlCount`

Number of dynamic flow control instructions used.

`MacroInstructionCount`

Number of macro instructions used.

`ArrayInstructionCount`

Number of array instructions used.

`CutInstructionCount`

Number of cut instructions used.

`EmitInstructionCount`

Number of emit instructions used.

`GSOoutputTopology`

The [D3D_PRIMITIVE_TOPOLOGY](#)-typed value that represents the geometry shader output topology.

`GSMaxOutputVertexCount`

Geometry shader maximum output vertex count.

`InputPrimitive`

The [D3D_PRIMITIVE](#)-typed value that represents the input primitive for a geometry shader or hull shader.

`PatchConstantParameters`

Number of parameters in the patch-constant signature.

`cGSInstanceCount`

Number of geometry shader instances.

`cControlPoints`

Number of control points in the hull shader and domain shader.

`HSOutputPrimitive`

The [D3D_TESSELLATOR_OUTPUT_PRIMITIVE](#)-typed value that represents the tessellator output-primitive type.

`HSTexture`

The [D3D_TESSELLATOR_PARTITIONING](#)-typed value that represents the tessellator partitioning mode.

`TessellatorDomain`

The [D3D_TESSELLATOR_DOMAIN](#)-typed value that represents the tessellator domain.

`cBarrierInstructions`

Number of barrier instructions in a compute shader.

`cInterlockedInstructions`

Number of interlocked instructions in a compute shader.

`cTextureStoreInstructions`

Number of texture writes in a compute shader.

Remarks

A shader is written in HLSL and compiled into an intermediate language by the HLSL compiler. The shader description returns information about the compiled shader. To get a shader description, call [ID3D12ShaderReflection::GetDesc](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_INPUT_BIND_DESC structure (d3d12shader.h)

Article 02/22/2024

Describes how a shader resource is bound to a shader input.

Syntax

C++

```
typedef struct _D3D12_SHADER_INPUT_BIND_DESC {
    LPCSTR             Name;
    D3D_SHADER_INPUT_TYPE Type;
    UINT               BindPoint;
    UINT               BindCount;
    UINT               uFlags;
    D3D_RESOURCE_RETURN_TYPE ReturnType;
    D3D_SRV_DIMENSION Dimension;
    UINT               NumSamples;
    UINT               Space;
    UINT               uID;
} D3D12_SHADER_INPUT_BIND_DESC;
```

Members

Name

Name of the shader resource.

Type

A [D3D_SHADER_INPUT_TYPE](#)-typed value that identifies the type of data in the resource.

BindPoint

Starting bind point.

BindCount

Number of contiguous bind points for arrays.

uFlags

A combination of [D3D_SHADER_INPUT_FLAGS](#)-typed values for shader input-parameter options.

ReturnType

If the input is a texture, the [D3D_RESOURCE_RETURN_TYPE](#)-typed value that identifies the return type.

Dimension

A [D3D_SRV_DIMENSION](#)-typed value that identifies the dimensions of the bound resource.

NumSamples

The number of samples for a multisampled texture; when a texture isn't multisampled, the value is set to -1 (0xFFFFFFFF). This is zero if the shader resource is not a recognized texture. If the shader resource is a structured buffer, the field contains the stride of the type in bytes.

Space

The register space.

uID

The range ID in the bytecode.

Remarks

Get a shader-input-signature description by calling [ID3D12ShaderReflection::GetResourceBindingDesc](#) or [ID3D12ShaderReflection::GetResourceBindingDescByName](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Header	d3d12shader.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_TYPE_DESC structure (d3d12shader.h)

Article 07/27/2022

Describes a shader-variable type.

Syntax

C++

```
typedef struct _D3D12_SHADER_TYPE_DESC {
    D3D_SHADER_VARIABLE_CLASS Class;
    D3D_SHADER_VARIABLE_TYPE Type;
    UINT Rows;
    UINT Columns;
    UINT Elements;
    UINT Members;
    UINT Offset;
    LPCSTR Name;
} D3D12_SHADER_TYPE_DESC;
```

Members

Class

A [D3D_SHADER_VARIABLE_CLASS](#)-typed value that identifies the variable class as one of scalar, vector, matrix, object, and so on.

Type

A [D3D_SHADER_VARIABLE_TYPE](#)-typed value that identifies the variable type.

Rows

Number of rows in a matrix. Otherwise a numeric type returns 1, any other type returns 0.

Columns

Number of columns in a matrix. Otherwise a numeric type returns 1, any other type returns 0.

Elements

Number of elements in an array; otherwise 0.

Members

Number of members in the structure; otherwise 0.

Offset

Offset, in bytes, between the start of the parent structure and this variable. Can be 0 if not a structure member.

Name

Name of the shader-variable type. This member can be **NULL** if it isn't used. This member supports dynamic shader linkage interface types, which have names. For more info about dynamic shader linkage, see [Dynamic Linking](#).

Remarks

Get a shader-variable-type description by calling [ID3D12ShaderReflectionType::GetDesc](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_VARIABLE_DESC structure (d3d12shader.h)

Article02/22/2024

Describes a shader variable.

Syntax

C++

```
typedef struct _D3D12_SHADER_VARIABLE_DESC {
    LPCSTR Name;
    UINT StartOffset;
    UINT Size;
    UINT uFlags;
    LPVOID DefaultValue;
    UINT StartTexture;
    UINT TextureSize;
    UINT StartSampler;
    UINT SamplerSize;
} D3D12_SHADER_VARIABLE_DESC;
```

Members

Name

The variable name.

StartOffset

Offset from the start of the parent structure to the beginning of the variable.

Size

Size of the variable (in bytes).

uFlags

A combination of [D3D_SHADER_VARIABLE_FLAGS](#)-typed values that are combined by using a bitwise-OR operation. The resulting value identifies shader-variable properties.

DefaultValue

The default value for initializing the variable. Emits default values for reflection.

StartTexture

Offset from the start of the variable to the beginning of the texture.

TextureSize

The size of the texture, in bytes.

StartSampler

Offset from the start of the variable to the beginning of the sampler.

SamplerSize

The size of the sampler, in bytes.

Remarks

Get a shader-variable description using reflection by calling [ID3D12ShaderReflectionVariable::GetDesc](#).

Requirements

 Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SHADER_VERSION_TYPE enumeration (d3d12shader.h)

Article 02/22/2024

Enumerates the types of shaders that Direct3D recognizes.

Used to encode the **Version** member of the [D3D12_SHADER_DESC](#) structure.

Syntax

C++

```
typedef enum D3D12_SHADER_VERSION_TYPE {
    D3D12_SHVER_PIXEL_SHADER = 0,
    D3D12_SHVER_VERTEX_SHADER = 1,
    D3D12_SHVER_GEOMETRY_SHADER = 2,
    D3D12_SHVER_HULL_SHADER = 3,
    D3D12_SHVER_DOMAIN_SHADER = 4,
    D3D12_SHVER_COMPUTE_SHADER = 5,
    D3D12_SHVER_LIBRARY,
    D3D12_SHVER_RAY_GENERATION_SHADER,
    D3D12_SHVER_INTERSECTION_SHADER,
    D3D12_SHVER_ANY_HIT_SHADER,
    D3D12_SHVER_CLOSEST_HIT_SHADER,
    D3D12_SHVER_MISS_SHADER,
    D3D12_SHVER_CALLABLE_SHADER,
    D3D12_SHVER_MESH_SHADER,
    D3D12_SHVER_AMPLIFICATION_SHADER,
    D3D12_SHVER_RESERVED0 = 0xFFFF
} ;
```

Constants

 [Expand table](#)

<code>D3D12_SHVER_PIXEL_SHADER</code>

Value: 0

Pixel shader.

<code>D3D12_SHVER_VERTEX_SHADER</code>
--

Value: 1

Vertex shader.

`D3D12_SHVER_GEOMETRY_SHADER`

Value: 2

Geometry shader.

`D3D12_SHVER_HULL_SHADER`

Value: 3

Hull shader.

`D3D12_SHVER_DOMAIN_SHADER`

Value: 4

Domain shader.

`D3D12_SHVER_COMPUTE_SHADER`

Value: 5

Compute shader.

`D3D12_SHVER_RESERVED0`

Value: *0xFFFF*

Indicates the end of the enumeration.

[+] Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Enumerations](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

D3D12_SIGNATURE_PARAMETER_DESC structure (d3d12shader.h)

Article 07/27/2022

Describes a shader signature.

Syntax

C++

```
typedef struct _D3D12_SIGNATURE_PARAMETER_DESC {
    LPCSTR SemanticName;
    UINT SemanticIndex;
    UINT Register;
    D3D_NAME SystemValueType;
    D3D_REGISTER_COMPONENT_TYPE ComponentType;
    BYTE Mask;
    BYTE ReadWriteMask;
    UINT Stream;
    D3D_MIN_PRECISION MinPrecision;
} D3D12_SIGNATURE_PARAMETER_DESC;
```

Members

SemanticName

A per-parameter string that identifies how the data will be used. For more info, see [Semantics](#).

SemanticIndex

Semantic index that modifies the semantic. Used to differentiate different parameters that use the same semantic.

Register

The register that will contain this variable's data.

SystemValueType

A [D3D_NAME](#)-typed value that identifies a predefined string that determines the functionality of certain pipeline stages.

ComponentType

A [D3D_REGISTER_COMPONENT_TYPE](#)-typed value that identifies the per-component-data type that is stored in a register. Each register can store up to four-components of data.

Mask

Mask which indicates which components of a register are used.

ReadWriteMask

Mask which indicates whether a given component is never written (if the signature is an output signature) or always read (if the signature is an input signature).

Stream

Indicates which stream the geometry shader is using for the signature parameter.

MinPrecision

A [D3D_MIN_PRECISION](#)-typed value that indicates the minimum desired interpolation precision. For more info, see [Using HLSL minimum precision](#).

Remarks

A shader can take n inputs and can produce m outputs. The order of the input (or output) parameters, their associated types, and any attached semantics make up the shader signature. Each shader has an input and an output signature.

When compiling a shader or an effect, some API calls validate shader signatures. That is, they compare the output signature of one shader (like a vertex shader) with the input signature of another shader (like a pixel shader). This ensures that a shader outputs data that is compatible with a downstream shader that is consuming that data. Compatible means that a shader signature is an exact-match subset of the preceding shader stage. Exact match means parameter types and semantics must exactly match. Subset means that a parameter that is not required by a downstream stage, does not need to include that parameter in its shader signature.

Get a shader-signature from a shader or an effect by calling APIs such as [ID3D12ShaderReflection::GetInputParameterDesc](#).

Requirements

Expand table

Requirement	Value
Header	d3d12shader.h

See also

[Shader Structures](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionParameterReflection interface (d3d12shader.h)

Article02/22/2024

A function-parameter-reflection interface accesses function-parameter info.

Note This interface is part of the HLSL shader linking technology that you can use on all Direct3D 12 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

Methods

The **ID3D12FunctionParameterReflection** interface has these methods.

[+] Expand table

[ID3D12FunctionParameterReflection::GetDesc](#)

Fills the parameter descriptor structure for the function's parameter.
([ID3D12FunctionParameterReflection.GetDesc](#))

Remarks

To get a function-parameter-reflection interface, call [ID3D12FunctionReflection::GetFunctionParameter](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

Note **ID3D12FunctionParameterReflection** requires the D3dcompiler_47.dll or a later version of the DLL.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionParameterReflection::GetDesc method (d3d12shader.h)

Article 02/22/2024

Fills the parameter descriptor structure for the function's parameter.

Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D12_PARAMETER_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [D3D12_PARAMETER_DESC*](#)

A pointer to a [D3D12_PARAMETER_DESC](#) structure that receives a description of the function's parameter.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionParameterReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection interface (d3d12shader.h)

Article 08/19/2022

A function-reflection interface accesses function info.

Note This interface is part of the HLSL shader linking technology that you can use on all Direct3D 12 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

Methods

The **ID3D12FunctionReflection** interface has these methods.

[+] Expand table

ID3D12FunctionReflection::GetConstantBufferByIndex
The <code>ID3D12FunctionReflection::GetConstantBufferByIndex</code> method (<code>d3d12shader.h</code>) gets a constant buffer by index for a function.
ID3D12FunctionReflection::GetConstantBufferByName
Gets a constant buffer by name for a function. (<code>ID3D12FunctionReflection.GetConstantBufferByName</code>)
ID3D12FunctionReflection::GetDesc
Fills the function descriptor structure for the function. (<code>ID3D12FunctionReflection.GetDesc</code>)
ID3D12FunctionReflection::GetFunctionParameter
Gets the function parameter reflector. (<code>ID3D12FunctionReflection.GetFunctionParameter</code>)
ID3D12FunctionReflection::GetResourceBindingDesc
Gets a description of how a resource is bound to a function. (<code>ID3D12FunctionReflection.GetResourceBindingDesc</code>)

[ID3D12FunctionReflection::GetResourceBindingDescByName](#)

Gets a description of how a resource is bound to a function.
(ID3D12FunctionReflection.GetResourceBindingDescByName)

[ID3D12FunctionReflection::GetVariableByName](#)

Gets a variable by name. (ID3D12FunctionReflection.GetVariableByName)

Remarks

To get a function-reflection interface, call [ID3D12LibraryReflection::GetFunctionByIndex](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

Note `ID3D12FunctionReflection` requires the `D3dcompiler_47.dll` or a later version of the DLL.

Requirements

[\[\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	<code>d3d12shader.h</code>

See also

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

ID3D12FunctionReflection::GetConstantBufferByIndex method (d3d12shader.h)

Article 02/22/2024

Gets a constant buffer by index for a function.

Syntax

C++

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    [in] UINT BufferIndex  
);
```

Parameters

[in] BufferIndex

Type: [UINT](#)

Zero-based index.

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

A pointer to a [ID3D12ShaderReflectionConstantBuffer](#) interface that represents the constant buffer.

Remarks

A constant buffer supplies either scalar constants or texture constants to a shader. A shader can use one or more constant buffers. For best performance, separate constants into buffers based on the frequency they are updated.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetConstantBufferByName method (d3d12shader.h)

Article 02/22/2024

Gets a constant buffer by name for a function.

Syntax

C++

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    [in] LPCSTR Name  
);
```

Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name.

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

A pointer to a [ID3D12ShaderReflectionConstantBuffer](#) interface that represents the constant buffer.

Remarks

A constant buffer supplies either scalar constants or texture constants to a shader. A shader can use one or more constant buffers. For best performance, separate constants into buffers based on the frequency they are updated.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetDesc method (d3d12shader.h)

Article 02/22/2024

Fills the function descriptor structure for the function.

Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D12_FUNCTION_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [D3D12_FUNCTION_DESC*](#)

A pointer to a [D3D12_FUNCTION_DESC](#) structure that receives a description of the function.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetFunctionParameter method (d3d12shader.h)

Article 02/22/2024

Gets the function parameter reflector.

Syntax

C++

```
ID3D12FunctionParameterReflection * GetFunctionParameter(
    [in] INT ParameterIndex
);
```

Parameters

[in] ParameterIndex

Type: **INT**

The zero-based index of the function parameter reflector to retrieve.

Return value

Type: **ID3D12FunctionParameterReflection***

A pointer to a **ID3D12FunctionParameterReflection** interface that represents the function parameter reflector.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetResourceBindingDesc method (d3d12shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a function.

Syntax

C++

```
HRESULT GetResourceBindingDesc(
    [in]  UINT             ResourceIndex,
    [out] D3D12_SHADER_INPUT_BIND_DESC *pDesc
);
```

Parameters

[in] ResourceIndex

Type: [UINT](#)

A zero-based resource index.

[out] pDesc

Type: [D3D12_SHADER_INPUT_BIND_DESC*](#)

A pointer to a [D3D12_SHADER_INPUT_BIND_DESC](#) structure that describes input binding of the resource.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDesc` gets info

about how one resource in the set is bound as an input to the shader. The *ResourceIndex* parameter specifies the index for the resource.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetResourceBindingDescByName method (d3d12shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a function.

Syntax

C++

```
HRESULT GetResourceBindingDescByName(  
    [in]    LPCSTR                 Name,  
    [out]   D3D12_SHADER_INPUT_BIND_DESC *pDesc  
) ;
```

Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

[out] pDesc

Type: [D3D12_SHADER_INPUT_BIND_DESC*](#)

A pointer to a [D3D12_SHADER_INPUT_BIND_DESC](#) structure that describes input binding of the resource.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDescByName` gets info about how one resource in the set is bound as an input to the shader. The *Name* parameter specifies the name of the resource.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12FunctionReflection::GetVariableByName method (d3d12shader.h)

Article 02/22/2024

Gets a variable by name.

Syntax

C++

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
) ;
```

Parameters

[in] Name

Type: [LPCSTR](#)

A pointer to a string containing the variable name.

Return value

Type: [ID3D12ShaderReflectionVariable*](#)

Returns a [ID3D12ShaderReflectionVariable Interface](#) interface.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12LibraryReflection interface (d3d12shader.h)

Article 02/22/2024

A library-reflection interface accesses library info.

Note This interface is part of the HLSL shader linking technology that you can use on all Direct3D 12 platforms to create precompiled HLSL functions, package them into libraries, and link them into full shaders at run time.

Inheritance

The **ID3D12LibraryReflection** interface inherits from the [IUnknown](#) interface.
ID3D12LibraryReflection also has these types of members:

Methods

The **ID3D12LibraryReflection** interface has these methods.

[] [Expand table](#)

ID3D12LibraryReflection::GetDesc
Fills the library descriptor structure for the library reflection. (ID3D12LibraryReflection.GetDesc)
ID3D12LibraryReflection::GetFunctionByIndex
The ID3D12LibraryReflection::GetFunctionByIndex method (d3d12shader.h) gets the function reflector.

Remarks

To get a library-reflection interface, call [D3DReflectLibrary](#).

Note ID3D12LibraryReflection requires the D3dcompiler_47.dll or a later version of the DLL.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[IUnknown](#)

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12LibraryReflection::GetDesc method (d3d12shader.h)

Article 02/22/2024

Fills the library descriptor structure for the library reflection.

Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D12_LIBRARY_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [D3D12_LIBRARY_DESC*](#)

A pointer to a [D3D12_LIBRARY_DESC](#) structure that receives a description of the library reflection.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12LibraryReflection](#)

Feedback

Was this page helpful?



[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12LibraryReflection::GetFunctionByIndex method (d3d12shader.h)

Article 02/22/2024

Gets the function reflector.

Syntax

C++

```
ID3D12FunctionReflection * GetFunctionByIndex(  
    [in] INT FunctionIndex  
);
```

Parameters

[in] FunctionIndex

Type: **INT**

The zero-based index of the function reflector to retrieve.

Return value

Type: **ID3D12FunctionReflection***

The function reflector, as a pointer to [ID3D12FunctionReflection](#).

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection interface (d3d12shader.h)

Article 07/27/2022

A shader-reflection interface accesses shader information.

Inheritance

The **ID3D12ShaderReflection** interface inherits from the [IUnknown](#) interface.

ID3D12ShaderReflection also has these types of members:

Methods

The **ID3D12ShaderReflection** interface has these methods.

 [Expand table](#)

ID3D12ShaderReflection::GetBitwiseInstructionCount
Gets the number of bitwise instructions. (<code>ID3D12ShaderReflection.GetBitwiseInstructionCount</code>)
ID3D12ShaderReflection::GetConstantBufferByIndex
Gets a constant buffer by index.
ID3D12ShaderReflection::GetConstantBufferByName
Gets a constant buffer by name.
ID3D12ShaderReflection::GetConversionInstructionCount
Gets the number of conversion instructions. (<code>ID3D12ShaderReflection.GetConversionInstructionCount</code>)
ID3D12ShaderReflection::GetDesc
Gets a shader description.
ID3D12ShaderReflection::GetGSInputPrimitive
Gets the geometry-shader input-primitive description. (<code>ID3D12ShaderReflection.GetGSInputPrimitive</code>)

ID3D12ShaderReflection::GetInputParameterDesc	Gets an input-parameter description for a shader.
ID3D12ShaderReflection::GetMinFeatureLevel	Gets the minimum feature level. (ID3D12ShaderReflection.GetMinFeatureLevel)
ID3D12ShaderReflection::GetMovcInstructionCount	Gets the number of Movc instructions. (ID3D12ShaderReflection.GetMovcInstructionCount)
ID3D12ShaderReflection::GetMovInstructionCount	Gets the number of Mov instructions. (ID3D12ShaderReflection.GetMovInstructionCount)
ID3D12ShaderReflection::GetNumInterfaceSlots	Gets the number of interface slots in a shader. (ID3D12ShaderReflection.GetNumInterfaceSlots)
ID3D12ShaderReflection::GetOutputParameterDesc	Gets an output-parameter description for a shader.
ID3D12ShaderReflection::GetPatchConstantParameterDesc	Gets a patch-constant parameter description for a shader.
ID3D12ShaderReflection::GetRequiresFlags	Gets a group of flags that indicates the requirements of a shader. (ID3D12ShaderReflection.GetRequiresFlags)
ID3D12ShaderReflection::GetResourceBindingDesc	Gets a description of how a resource is bound to a shader. (ID3D12ShaderReflection.GetResourceBindingDesc)
ID3D12ShaderReflection::GetResourceBindingDescByName	Gets a description of how a resource is bound to a shader. (ID3D12ShaderReflection.GetResourceBindingDescByName)
ID3D12ShaderReflection::GetThreadGroupSize	Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid. (ID3D12ShaderReflection.GetThreadGroupSize)

[ID3D12ShaderReflection::GetVariableByName](#)

Gets a variable by name. ([ID3D12ShaderReflection.GetVariableByName](#))

[ID3D12ShaderReflection::IsSampleFrequencyShader](#)

Indicates whether a shader is a sample frequency shader.
([ID3D12ShaderReflection.IsSampleFrequencyShader](#))

Remarks

An [ID3D12ShaderReflection](#) interface can be retrieved for a shader by using [D3DReflect](#).

Note

This function from `d3dcompiler.dll` supports Shader Model 2 - 5.1. For Shader Model 6 shader reflection, see `dxcompiler.dll` and [Using dxc.exe and dxcompiler.dll](#).

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	<code>d3d12shader.h</code>

See also

[IUnknown](#)

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12ShaderReflection::GetBitwiseInstructionCount method (d3d12shader.h)

Article 02/22/2024

Gets the number of bitwise instructions.

Syntax

C++

```
UINT GetBitwiseInstructionCount();
```

Return value

Type: [UINT](#)

The number of bitwise instructions.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetConstantBufferByIndex method (d3d12shader.h)

Article 02/22/2024

Gets a constant buffer by index.

Syntax

C++

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    [in] UINT Index  
);
```

Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

A pointer to a constant buffer (see [ID3D12ShaderReflectionConstantBuffer Interface](#)).

Remarks

A constant buffer supplies either scalar constants or texture constants to a shader. A shader can use one or more constant buffers. For best performance, separate constants into buffers based on the frequency they are updated.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetConstantBufferByName method (d3d12shader.h)

Article 10/13/2021

Gets a constant buffer by name.

Syntax

C++

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    [in] LPCSTR Name  
);
```

Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name.

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

A pointer to a constant buffer (see [ID3D12ShaderReflectionConstantBuffer Interface](#)).

Remarks

A constant buffer supplies either scalar constants or texture constants to a shader. A shader can use one or more constant buffers. For best performance, separate constants into buffers based on the frequency they are updated.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetConversionInstructionCount method (d3d12shader.h)

Article 02/22/2024

Gets the number of conversion instructions.

Syntax

C++

```
UINT GetConversionInstructionCount();
```

Return value

Type: [UINT](#)

Returns the number of conversion instructions.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetDesc method (d3d12shader.h)

Article 02/22/2024

Gets a shader description.

Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D12_SHADER_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [D3D12_SHADER_DESC*](#)

A shader description, as a pointer to a [D3D12_SHADER_DESC](#) structure.

Return value

Type: [HRESULT](#)

Returns one of the following [Direct3D 12 Return Codes](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflection::GetGSInputPrimitive method (d3d12shader.h)

Article 02/22/2024

Gets the geometry-shader input-primitive description.

Syntax

C++

```
D3D_PRIMITIVE GetGSInputPrimitive();
```

Return value

Type: [D3D_PRIMITIVE](#)

The input-primitive description. See [D3D_PRIMITIVE_TOPOLOGY](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetInputParameterDesc method (d3d12shader.h)

Article 02/22/2024

Gets an input-parameter description for a shader.

Syntax

C++

```
HRESULT GetInputParameterDesc(  
    [in]  UINT             ParameterIndex,  
    [out] D3D12_SIGNATURE_PARAMETER_DESC *pDesc  
);
```

Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D12_SIGNATURE_PARAMETER_DESC*](#)

A pointer to a shader-input-signature description. See [D3D12_SIGNATURE_PARAMETER_DESC](#).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

An input-parameter description is also called a shader signature. The shader signature contains information about the input parameters such as the order or parameters, their

data type, and a parameter semantic.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetMinFeatureLevel method (d3d12shader.h)

Article 02/22/2024

Gets the minimum feature level.

Syntax

C++

```
HRESULT GetMinFeatureLevel(
    [out] D3D_FEATURE_LEVEL *pLevel
);
```

Parameters

[out] pLevel

Type: [D3D_FEATURE_LEVEL*](#)

A pointer to one of the enumerated values in [D3D_FEATURE_LEVEL](#), which represents the minimum feature level.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetMovcInstructionCount method (d3d12shader.h)

Article 02/22/2024

Gets the number of Movc instructions.

Syntax

C++

```
UINT GetMovcInstructionCount();
```

Return value

Type: [UINT](#)

Returns the number of Movc instructions.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetMovInstructionCount method (d3d12shader.h)

Article 02/22/2024

Gets the number of Mov instructions.

Syntax

C++

```
UINT GetMovInstructionCount();
```

Return value

Type: [UINT](#)

Returns the number of Mov instructions.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetNumInterfaceSlots method (d3d12shader.h)

Article 02/22/2024

Gets the number of interface slots in a shader.

Syntax

C++

```
UINT GetNumInterfaceSlots();
```

Return value

Type: [UINT](#)

The number of interface slots in the shader.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetOutputParameterDesc method (d3d12shader.h)

Article 02/22/2024

Gets an output-parameter description for a shader.

Syntax

C++

```
HRESULT GetOutputParameterDesc(
    [in]    UINT           ParameterIndex,
    [out]   D3D12_SIGNATURE_PARAMETER_DESC *pDesc
);
```

Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D12_SIGNATURE_PARAMETER_DESC*](#)

A shader-output-parameter description, as a pointer to a [D3D12_SIGNATURE_PARAMETER_DESC](#) structure.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

An output-parameter description is also called a shader signature. The shader signature contains information about the output parameters such as the order or parameters,

their data type, and a parameter semantic.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetPatchConstantParameterDesc method (d3d12shader.h)

Article 02/22/2024

Gets a patch-constant parameter description for a shader.

Syntax

C++

```
HRESULT GetPatchConstantParameterDesc(
    [in]    UINT           ParameterIndex,
    [out]   D3D12_SIGNATURE_PARAMETER_DESC *pDesc
);
```

Parameters

[in] ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

[out] pDesc

Type: [D3D12_SIGNATURE_PARAMETER_DESC*](#)

A pointer to a shader-input-signature description. See [D3D12_SIGNATURE_PARAMETER_DESC](#).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetRequiresFlags method (d3d12shader.h)

Article 06/09/2023

Retrieves a group of flags that indicate the requirements of a shader.

Syntax

C++

```
UINT64 GetRequiresFlags();
```

Return value

Type: **UINT64**

A value that contains a combination of one or more shader requirements `#define` flags; each flag specifies a requirement of the shader. A default value of 0 means that there are no requirements.

D3D_SHADER_REQUIRES_DOUBLES. Shader requires that the graphics driver and hardware support the *double* data type.

D3D_SHADER_REQUIRES_EARLY_DEPTH_STENCIL. Shader requires an early depth stencil.

D3D_SHADER_REQUIRES_UAVS_AT_EVERY_STAGE. Shader requires unordered access views (UAVs) at every pipeline stage.

D3D_SHADER_REQUIRES_64_UAVS. Shader requires 64 UAVs.

D3D_SHADER_REQUIRES_MINIMUM_PRECISION. Shader requires the graphics driver and hardware to support minimum precision. For more info, see [Using HLSL minimum precision](#).

D3D_SHADER_REQUIRES_11_1_DOUBLE_EXTENSIONS. Shader requires that the graphics driver and hardware support extended doubles instructions. For more info, see the `ExtendedDoublesShaderInstructions` member of [D3D12_FEATURE_DATA_D3D12_OPTIONS](#).

D3D_SHADER_REQUIRES_11_1_SHADER_EXTENSIONS. Shader requires that the graphics driver and hardware support the [msad4](#) intrinsic function in shaders. For more info, see the **SAD4ShaderInstructions** member of [D3D12_FEATURE_DATA_D3D12_OPTIONS](#).

D3D_SHADER_REQUIRES_LEVEL_9_COMPARISON_FILTERING. Shader requires that the graphics driver and hardware support Direct3D 9 shadow support.

D3D_SHADER_REQUIRES_TILED_RESOURCES. Shader requires that the graphics driver and hardware support tiled resources.

D3D_SHADER_REQUIRES_STENCIL_REF. Shader requires a reference value for depth stencil tests. For more info, see the **PSSpecifiedStencilRefSupported** member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure, and [ID3D12GraphicsCommandList::OMSetStencilRef](#).

D3D_SHADER_REQUIRES_INNER_COVERAGE. Shader requires that the graphics driver and hardware support inner coverage. For more info, see the enumeration constants **D3D_NAME_INNER_COVERAGE** and **D3D11_NAME_INNER_COVERAGE** in [D3D_NAME](#).

D3D_SHADER_REQUIRES_TYPED_UAV_LOAD_ADDITIONAL_FORMATS. Shader requires that the graphics driver and hardware support the loading of additional formats for typed unordered-access views (UAVs). See the *TypedUAVLoadAdditionalFormats* member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

D3D_SHADER_REQUIRES_ROVS. Shader requires that the graphics driver and hardware support rasterizer ordered views (ROVs). See [Rasterizer Ordered Views](#).

D3D_SHADER_REQUIRES_VIEWPORT_AND_RT_ARRAY_INDEX_FROM_ANY_SHADER_FEEDING_RASTERIZER. Shader requires that the graphics driver and hardware support viewport and render target array index values from any shader-feeding rasterizer. For more info, see the member *VPAAndRTArrayIndexFromAnyShaderFeedingRasterizerSupportedWithoutGSEmulation* of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

D3D_SHADER_REQUIRES_WAVE_OPS. Shader requires that the graphics driver and hardware support wave ops. For more info, see the member **WaveOps** of the [D3D12_FEATURE_DATA_D3D12_OPTIONS1](#) structure.

D3D_SHADER_REQUIRES_INT64_OPS. Shader requires that the graphics driver and hardware support 64-bit integer ops. For more info, see the member **Int64ShaderOps** of the [D3D12_FEATURE_DATA_D3D12_OPTIONS1](#) structure.

D3D_SHADER_REQUIRES_VIEW_ID. Shader requires that the graphics driver and hardware support view instancing using **SV_ViewID**. For more info, see the member

`ViewInstancingTier` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS3](#) structure.

D3D_SHADER_REQUIRES_BARYCENTRICS. Shader requires that the graphics driver and hardware support barycentrics using `SV_Barycentrics`. For more info, see the member `BarycentricsSupported` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS3](#) structure.

D3D_SHADER_REQUIRES_NATIVE_16BIT_OPS. Shader requires that the graphics driver and hardware support native 16-bit ops. For more info, see the member `Native16BitShaderOpsSupported` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS4](#) structure.

D3D_SHADER_REQUIRES_SHADING_RATE. Shader requires that the graphics driver and hardware support the Variable Shading Rate (VRS) feature. For more info, see the member `VariableShadingRateTier` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS6](#) structure.

D3D_SHADER_REQUIRES_RAYTRACING_TIER_1_1. Shader requires that the graphics driver and hardware support DXR tier 1.1. For more info, see the member `RaytracingTier` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) structure.

D3D_SHADER_REQUIRES_SAMPLER_FEEDBACK. Shader requires that the graphics driver and hardware support Sampler Feedback. For more info, see the member `SamplerFeedbackTier` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS7](#) structure.

D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_TYPED_RESOURCE. Shader requires that the graphics driver and hardware support int64 atomics on typed resources. For more info, see the member `AtomicInt64OnTypedResourceSupported` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS9](#) structure.

D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_GROUP_SHARED. Shader requires that the graphics driver and hardware support int64 atomics on groupshared memory. For more info, see the member `AtomicInt64OnGroupSharedSupported` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS9](#) structure.

D3D_SHADER_REQUIRES_DERIVATIVES_IN_MESH_AND_AMPLIFICATION_SHADERS. Shader requires that the graphics driver and hardware support derivatives in mesh and amplification shaders. For more info, see the member `DerivativesInMeshAndAmplificationShadersSupported` of the [D3D12_FEATURE_DATA_D3D12_OPTIONS9](#) structure.

D3D_SHADER_REQUIRES_RESOURCE_DESCRIPTOR_HEAP_INDEXING. Shader requires that the graphics driver and hardware support Dynamic Resources (a requirement for Shader Model 6.6) and the `ResourceDescriptorHeap` in particular. For more info, see the [HLSL dynamic resources](#) spec on GitHub.

D3D_SHADER_REQUIRES_SAMPLER_DESCRIPTOR_HEAP_INDEXING. Shader requires that the graphics driver and hardware support Dynamic Resources (a requirement for Shader Model 6.6) and the **SamplerDescriptorHeap** in particular. For more info, see the [HLSL dynamic resources](#) spec on GitHub.

D3D_SHADER_REQUIRES_WAVE_MMA. Shader requires that the graphics driver and hardware support Wave MMA. For more info, see the member **WaveMMATier** of the **D3D12_FEATURE_DATA_D3D12_OPTIONS9** structure.

D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_DESCRIPTOR_HEAP_RESOURCE. Shader requires that the graphics driver and hardware support int64 atomics on descriptor heap resources. For more info, see the member **AtomicInt64OnDescriptorHeapResourceSupported** of the **D3D12_FEATURE_DATA_D3D12_OPTIONS11** structure.

Remarks

Here's how the `d3d12shader.h` header file defines the shader requirements flags:

C++

```
#define D3D_SHADER_REQUIRES_DOUBLES  
0x00000001  
#define D3D_SHADER_REQUIRES_EARLY_DEPTH_STENCIL  
0x00000002  
#define D3D_SHADER_REQUIRES_UAVS_AT_EVERY_STAGE  
0x00000004  
#define D3D_SHADER_REQUIRES_64_UAVS  
0x00000008  
#define D3D_SHADER_REQUIRES_MINIMUM_PRECISION  
0x00000010  
#define D3D_SHADER_REQUIRES_11_1_DOUBLE_EXTENSIONS  
0x00000020  
#define D3D_SHADER_REQUIRES_11_1_SHADER_EXTENSIONS  
0x00000040  
#define D3D_SHADER_REQUIRES_LEVEL_9_COMPARISON_FILTERING  
0x00000080  
#define D3D_SHADER_REQUIRES_TILED_RESOURCES  
0x00000100  
#define D3D_SHADER_REQUIRES_STENCIL_REF  
0x00000200  
#define D3D_SHADER_REQUIRES_INNER_COVERAGE  
0x00000400  
#define D3D_SHADER_REQUIRES_TYPED_UAV_LOAD_ADDITIONAL_FORMATS  
0x00000800  
#define D3D_SHADER_REQUIRES_ROVS  
0x00001000  
#define  
D3D_SHADER_REQUIRES_VIEWPORT_AND_RT_ARRAY_INDEX_FROM_ANY_SHADER_FEEDING_RAST
```

```
ERIZER 0x00002000
#define D3D_SHADER_REQUIRES_WAVE_OPS
0x00004000
#define D3D_SHADER_REQUIRES_INT64_OPS
0x00008000
#define D3D_SHADER_REQUIRES_VIEW_ID
0x00010000
#define D3D_SHADER_REQUIRES_BARYCENTRICS
0x00020000
#define D3D_SHADER_REQUIRES_NATIVE_16BIT_OPS
0x00040000
#define D3D_SHADER_REQUIRES_SHADING_RATE
0x00080000
#define D3D_SHADER_REQUIRES_RAYTRACING_TIER_1_1
0x00100000
#define D3D_SHADER_REQUIRES_SAMPLER_FEEDBACK
0x00200000
#define D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_TYPED_RESOURCE
0x00400000
#define D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_GROUP_SHARED
0x00800000
#define D3D_SHADER_REQUIRES_DERIVATIVES_IN_MESH_AND_AMPLIFICATION_SHADERS
0x01000000
#define D3D_SHADER_REQUIRES_RESOURCE_DESCRIPTOR_HEAP_INDEXING
0x02000000
#define D3D_SHADER_REQUIRES_SAMPLER_DESCRIPTOR_HEAP_INDEXING
0x04000000
#define D3D_SHADER_REQUIRES_WAVE_MMA
0x08000000
#define D3D_SHADER_REQUIRES_ATOMIC_INT64_ON_DESCRIPTOR_HEAP_RESOURCE
0x10000000
```

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

- [Checking hardware feature support](#)
- [ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetResourceBindingDesc method (d3d12shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a shader.

Syntax

C++

```
HRESULT GetResourceBindingDesc(
    [in]  UINT             ResourceIndex,
    [out] D3D12_SHADER_INPUT_BIND_DESC *pDesc
);
```

Parameters

[in] ResourceIndex

Type: [UINT](#)

A zero-based resource index.

[out] pDesc

Type: [D3D12_SHADER_INPUT_BIND_DESC*](#)

A pointer to an input-binding description. See [D3D12_SHADER_INPUT_BIND_DESC](#).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. **GetResourceBindingDesc** gets

information about how one resource in the set is bound as an input to the shader. The *ResourceIndex* parameter specifies the index for the resource.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetResourceBindingDescByName method (d3d12shader.h)

Article 02/22/2024

Gets a description of how a resource is bound to a shader.

Syntax

C++

```
HRESULT GetResourceBindingDescByName(  
    [in]    LPCSTR                 Name,  
    [out]   D3D12_SHADER_INPUT_BIND_DESC *pDesc  
) ;
```

Parameters

[in] Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

[out] pDesc

Type: [D3D12_SHADER_INPUT_BIND_DESC*](#)

A pointer to an input-binding description. See [D3D12_SHADER_INPUT_BIND_DESC](#).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

A shader consists of executable code (the compiled HLSL functions) and a set of resources that supply the shader with input data. `GetResourceBindingDescByName` gets information about how one resource in the set is bound as an input to the shader. The *Name* parameter specifies the name of the resource.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetThreadGroupSize method (d3d12shader.h)

Article 02/22/2024

Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid.

Syntax

C++

```
UINT GetThreadGroupSize(  
    [out, optional] UINT *pSizeX,  
    [out, optional] UINT *pSizeY,  
    [out, optional] UINT *pSizeZ  
) ;
```

Parameters

[out, optional] pSizeX

Type: **UINT***

A pointer to the size, in threads, of the x-dimension of the thread-group grid. The maximum size is 1024.

[out, optional] pSizeY

Type: **UINT***

A pointer to the size, in threads, of the y-dimension of the thread-group grid. The maximum size is 1024.

[out, optional] pSizeZ

Type: **UINT***

A pointer to the size, in threads, of the z-dimension of the thread-group grid. The maximum size is 64.

Return value

Type: [UINT](#)

Returns the total size, in threads, of the thread-group grid by calculating the product of the size of each dimension.

syntax

```
*pSizeX * *pSizeY * *pSizeZ;
```

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

When a compute shader is written it defines the actions of a single thread group only. If multiple thread groups are required, it is the role of the [ID3D12GraphicsCommandList::Dispatch](#) call to issue multiple thread groups.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflection::GetVariableByName method (d3d12shader.h)

Article 02/22/2024

Gets a variable by name.

Syntax

C++

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
) ;
```

Parameters

[in] Name

Type: [LPCSTR](#)

A pointer to a string containing the variable name.

Return value

Type: [ID3D12ShaderReflectionVariable*](#)

Returns a [ID3D12ShaderReflectionVariable Interface](#) interface.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflection::IsSampleFrequencyShader method (d3d12shader.h)

Article 02/22/2024

Indicates whether a shader is a sample frequency shader.

Syntax

C++

```
BOOL IsSampleFrequencyShader();
```

Return value

Type: **BOOL**

Returns true if the shader is a sample frequency shader; otherwise returns false.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionConstantBuffer interface (d3d12shader.h)

Article 02/22/2024

This shader-reflection interface provides access to a constant buffer.

Methods

The **ID3D12ShaderReflectionConstantBuffer** interface has these methods.

[+] Expand table

ID3D12ShaderReflectionConstantBuffer::GetDesc
Gets a constant-buffer description.
ID3D12ShaderReflectionConstantBuffer::GetVariableByIndex
Gets a shader-reflection variable by index.
ID3D12ShaderReflectionConstantBuffer::GetVariableByName
Gets a shader-reflection variable by name.

Remarks

To create a constant-buffer interface, call

[ID3D12ShaderReflection::GetConstantBufferByIndex](#) or

[ID3D12ShaderReflection::GetConstantBufferByName](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

Requirements

[+] Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	d3d12shader.h

See also

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionConstantBuffer::GetDesc method (d3d12shader.h)

Article 02/22/2024

Gets a constant-buffer description.

Syntax

C++

```
HRESULT GetDesc(  
    D3D12_SHADER_BUFFER_DESC *pDesc  
);
```

Parameters

pDesc

Type: [D3D12_SHADER_BUFFER_DESC*](#)

A shader-buffer description, as a pointer to a [D3D12_SHADER_BUFFER_DESC](#) structure.

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionConstantBuffer::GetVariableByIndex method (d3d12shader.h)

Article 02/22/2024

Gets a shader-reflection variable by index.

Syntax

C++

```
ID3D12ShaderReflectionVariable * GetVariableByIndex(  
    [in] UINT Index  
);
```

Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

Return value

Type: [ID3D12ShaderReflectionVariable*](#)

A pointer to a shader-reflection variable interface (see [ID3D12ShaderReflectionVariable Interface](#)).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionConstantBuffer::GetVariableByName method (d3d12shader.h)

Article 02/22/2024

Gets a shader-reflection variable by name.

Syntax

C++

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    [in] LPCSTR Name  
)
```

Parameters

[in] Name

Type: [LPCSTR](#)

Variable name.

Return value

Type: [ID3D12ShaderReflectionVariable*](#)

Returns a sentinel object (end of list marker). To determine if GetVariableByName successfully completed, call [ID3D12ShaderReflectionVariable::GetDesc](#) and check the returned **HRESULT**; any return value other than success means that GetVariableByName failed.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionType interface (d3d12shader.h)

Article 07/27/2022

This shader-reflection interface provides access to variable type.

Methods

The **ID3D12ShaderReflectionType** interface has these methods.

[+] Expand table

ID3D12ShaderReflectionType::GetBaseClass
Gets an ID3D12ShaderReflectionType Interface interface containing the variable base class type.
ID3D12ShaderReflectionType::GetDesc
Gets the description of a shader-reflection-variable type.
ID3D12ShaderReflectionType::GetInterfaceByIndex
Gets an interface by index.
ID3D12ShaderReflectionType::GetMemberTypeByIndex
Gets a shader-reflection-variable type by index.
ID3D12ShaderReflectionType::GetMemberTypeByName
Gets a shader-reflection-variable type by name.
ID3D12ShaderReflectionType::GetMemberTypeName
Gets a shader-reflection-variable type.
ID3D12ShaderReflectionType::GetNumInterfaces
Gets the number of interfaces. (<code>ID3D12ShaderReflectionType.GetNumInterfaces</code>)
ID3D12ShaderReflectionType::GetSubType
Gets the base class of a class. (<code>ID3D12ShaderReflectionType.GetSubType</code>)

[ID3D12ShaderReflectionType::ImplementsInterface](#)

Indicates whether a class type implements an interface.
([ID3D12ShaderReflectionType::ImplementsInterface](#))

[ID3D12ShaderReflectionType::IsEqual](#)

Indicates whether two ID3D12ShaderReflectionType Interface pointers have the same underlying type.

[ID3D12ShaderReflectionType::IsOfType](#)

Indicates whether a variable is of the specified type. ([ID3D12ShaderReflectionType::IsOfType](#))

Remarks

The get a shader-reflection-type interface, call

[ID3D12ShaderReflectionVariable::GetType](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

Requirements

[Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetBaseClass method (d3d12shader.h)

Article 02/22/2024

Gets an [ID3D12ShaderReflectionType Interface](#) interface containing the variable base class type.

Syntax

C++

```
ID3D12ShaderReflectionType * GetBaseClass();
```

Return value

Type: [ID3D12ShaderReflectionType*](#)

Returns A pointer to a [ID3D12ShaderReflectionType Interface](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionType::GetDesc method (d3d12shader.h)

Article 02/22/2024

Gets the description of a shader-reflection-variable type.

Syntax

C++

```
HRESULT GetDesc(  
    [out] D3D12_SHADER_TYPE_DESC *pDesc  
);
```

Parameters

[out] pDesc

Type: [D3D12_SHADER_TYPE_DESC*](#)

A pointer to a shader-type description (see [D3D12_SHADER_TYPE_DESC](#)).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetInterfaceByIndex method (d3d12shader.h)

Article 02/22/2024

Gets an interface by index.

Syntax

C++

```
ID3D12ShaderReflectionType * GetInterfaceByIndex(  
    [in] UINT uIndex  
);
```

Parameters

[in] uIndex

Type: [UINT](#)

Zero-based index.

Return value

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType](#) Interface.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetMemberTypeByIndex method (d3d12shader.h)

Article 02/22/2024

Gets a shader-reflection-variable type by index.

Syntax

C++

```
ID3D12ShaderReflectionType * GetMemberTypeByIndex(  
    [in] UINT Index  
);
```

Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

Return value

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType](#) interface.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetMemberTypeByName method (d3d12shader.h)

Article 02/22/2024

Gets a shader-reflection-variable type by name.

Syntax

C++

```
ID3D12ShaderReflectionType * GetMemberTypeByName(  
    [in] LPCSTR Name  
>;
```

Parameters

[in] Name

Type: [LPCSTR](#)

Member name.

Return value

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType](#) interface.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[] Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetMemberTypeName method (d3d12shader.h)

Article 02/22/2024

Gets a shader-reflection-variable type.

Syntax

C++

```
LPCSTR GetMemberTypeName(  
    [in] UINT Index  
);
```

Parameters

[in] Index

Type: [UINT](#)

Zero-based index.

Return value

Type: [LPCSTR](#)

The variable type.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::GetNumInterfaces method (d3d12shader.h)

Article 02/22/2024

Gets the number of interfaces.

Syntax

C++

```
UINT GetNumInterfaces();
```

Return value

Type: [UINT](#)

Returns the number of interfaces.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionType::GetSubType method (d3d12shader.h)

Article 02/22/2024

Gets the base class of a class.

Syntax

C++

```
ID3D12ShaderReflectionType * GetSubType();
```

Return value

Type: [ID3D12ShaderReflectionType*](#)

Returns a pointer to an [ID3D12ShaderReflectionType](#) containing the base class type.

Returns **NULL** if the class does not have a base class.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionType::ImplementsInterface method (d3d12shader.h)

Article 02/22/2024

Indicates whether a class type implements an interface.

Syntax

C++

```
HRESULT ImplementsInterface(  
    [in] ID3D12ShaderReflectionType *pBase  
);
```

Parameters

[in] pBase

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType](#) Interface.

Return value

Type: [HRESULT](#)

Returns S_OK if the interface is implemented; otherwise return S_FALSE.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows

Requirement

Value shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | Get help at Microsoft Q&A

ID3D12ShaderReflectionType::IsEqual method (d3d12shader.h)

Article02/22/2024

Indicates whether two [ID3D12ShaderReflectionType Interface](#) pointers have the same underlying type.

Syntax

C++

```
HRESULT IsEqual(
    [in] ID3D12ShaderReflectionType *pType
);
```

Parameters

[in] pType

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType Interface](#).

Return value

Type: [HRESULT](#)

Returns S_OK if the pointers have the same underlying type; otherwise returns S_FALSE.

Remarks

IsEqual indicates whether the sources of the [ID3D12ShaderReflectionType Interface](#) pointers have the same underlying type. For example, if two [ID3D12ShaderReflectionType Interface](#) pointers were retrieved from variables, IsEqual can be used to see if the variables have the same type.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionType::IsOfType method (d3d12shader.h)

Article 02/22/2024

Indicates whether a variable is of the specified type.

Syntax

C++

```
HRESULT IsOfType(  
    [in] ID3D12ShaderReflectionType *pType  
);
```

Parameters

[in] pType

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType](#) interface.

Return value

Type: [HRESULT](#)

Returns S_OK if object being queried is equal to or inherits from the type in the *pType* parameter; otherwise returns S_FALSE.

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[\[+\] Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionVariable interface (d3d12shader.h)

Article02/22/2024

This shader-reflection interface provides access to a variable.

Methods

The **ID3D12ShaderReflectionVariable** interface has these methods.

[+] Expand table

ID3D12ShaderReflectionVariable::GetBuffer
Returns the ID3D12ShaderReflectionConstantBuffer of the present ID3D12ShaderReflectionVariable.
ID3D12ShaderReflectionVariable::GetDesc
Gets a shader-variable description.
ID3D12ShaderReflectionVariable::GetInterfaceSlot
Gets the corresponding interface slot for a variable that represents an interface pointer. (ID3D12ShaderReflectionVariable.GetInterfaceSlot)
ID3D12ShaderReflectionVariable::GetType
Gets a shader-variable type.

Remarks

To get a shader-reflection-variable interface, call a method like [ID3D12ShaderReflection::GetVariableByName](#). This isn't a COM interface, so you don't need to worry about reference counts or releasing the interface when you're done with it.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionVariable::GetBuffer method (d3d12shader.h)

Article 02/22/2024

Returns the [ID3D12ShaderReflectionConstantBuffer](#) of the present [ID3D12ShaderReflectionVariable](#).

Syntax

C++

```
ID3D12ShaderReflectionConstantBuffer * GetBuffer();
```

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

Returns a pointer to the [ID3D12ShaderReflectionConstantBuffer](#) of the present [ID3D12ShaderReflectionVariable](#).

Requirements

[] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

ID3D12ShaderReflectionVariable::GetDesc method (d3d12shader.h)

Article 02/22/2024

Gets a shader-variable description.

Syntax

C++

```
HRESULT GetDesc(
    [out] D3D12_SHADER_VARIABLE_DESC *pDesc
);
```

Parameters

[out] pDesc

Type: [D3D12_SHADER_VARIABLE_DESC*](#)

A pointer to a shader-variable description (see [D3D12_SHADER_VARIABLE_DESC](#)).

Return value

Type: [HRESULT](#)

Returns one of the [Direct3D 12 Return Codes](#).

Remarks

This method can be used to determine if the [ID3D12ShaderReflectionVariable Interface](#) is valid, the method returns [E_FAIL](#) when the variable is not valid.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionVariable::GetInt erfaceSlot method (d3d12shader.h)

Article 02/22/2024

Gets the corresponding interface slot for a variable that represents an interface pointer.

Syntax

C++

```
UINT GetInterfaceSlot(  
    [in] UINT uArrayIndex  
>;
```

Parameters

[in] uArrayIndex

Type: [UINT](#)

The index of the array element to get the slot number for. For a non-array variable this value will be zero.

Return value

Type: [UINT](#)

Returns the index of the interface in the interface array.

Remarks

GetInterfaceSlot gets the corresponding slot in a dynamic linkage array for an interface instance. The returned slot number is used to set an interface instance to a particular class instance. See the HLSL [Interfaces and Classes](#) overview for additional information.

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

 Expand table

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

ID3D12ShaderReflectionVariable::GetType method (d3d12shader.h)

Article 02/22/2024

Gets a shader-variable type.

Syntax

C++

```
ID3D12ShaderReflectionType * GetType();
```

Return value

Type: [ID3D12ShaderReflectionType*](#)

A pointer to a [ID3D12ShaderReflectionType Interface](#).

Remarks

This method's interface is hosted in the out-of-box DLL D3DCompiler_xx.dll.

Requirements

[+] [Expand table](#)

Requirement	Value
Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

windows.graphics.holographic.interop.h header

Article01/24/2023

The APIs in the `Windows.Graphics.Holographic.Interop.h` header allow Windows Mixed Reality apps to use Direct3D 12. The interfaces specified in this header use COM interface pointers to pass DirectX COM objects as parameters to methods on Windows Runtime objects in the `Windows.Graphics.Holographic` namespace, allowing Windows Mixed Reality apps to create and use Direct3D 12 buffer resources with no additional overhead.

Sample code for this API set is included in the [Windows Mixed Reality Direct3D 12 app template](#). The Windows Mixed Reality Direct3D 12 app template includes boilerplate code for most APIs that are provided in the `Windows.Graphics.Holographic.Interop.h` header, and renders a spinning cube on a Windows Mixed Reality PC, a HoloLens 2, and the HoloLens 2 emulator.

This header is used by Direct3D 12 Graphics. For more information, see:

- [Direct3D 12 Graphics](#)

`windows.graphics.holographic.interop.h` contains the following programming interfaces:

Interfaces

[+] [Expand table](#)

`graphics::holographic::interop::IHolographicCameraInterop`

Extends `HolographicCamera` to allow 2D texture resources to be created and used as back buffers for holographic rendering in Direct3D 12.

`graphics::holographic::interop::IHolographicCameraRenderingParametersInterop`

A nano-COM interface that allows COM interop with the `HolographicCameraRenderingParameters` class for applications that use Direct3D 12 for holographic rendering.

`graphics::holographic::interop::IHolographicQuadLayerInterop`

A nano-COM interface that allows COM interop with the `HolographicQuadLayer` Windows Runtime class for apps that use Direct3D 12 for holographic rendering.

`graphics::holographic::interop::IHolographicQuadLayerUpdateParametersInterop`

A nano-COM interface that allows COM interop with the [HolographicQuadLayerUpdateParameters](#) class for applications that use Direct3D 12 for holographic rendering.

IHolographicCameraInterop interface (windows.graphics.holographic.interop.h)

Article08/23/2022

The [IHolographicCameraInterop](#) interface is a nano-COM interface, used to create Direct3D 12 back buffer resources for a [HolographicCamera](#) Windows Runtime object. This is an initialization step for using Direct3D 12 with Windows Mixed Reality. This interface also allows your application to acquire ownership of content buffers for rendering, prior to committing them with the [HolographicCameraRenderingParametersInterop](#) interface.

Your application can use this interface to initialize holographic rendering using Direct3D 12. Nano-COM allows pointers to Direct3D 12 objects to be passed directly as parameters for API calls, instead of using a Windows Runtime container object.

Your application manages its own pool of holographic buffer resources for use as render targets (back buffers) for each [HolographicCamera](#). It can create additional buffers as needed in order to continue rendering smoothly. On most devices, this will be three or four surfaces. Your application should start with at least two buffers in the pool. Your application can dynamically detect when it needs to create a new buffer by looking for unsuccessful attempts to immediately acquire buffers that were previously committed for presentation. Your application must commit a buffer for a [HolographicCamera](#) that is included on the [HolographicFrame](#) unless the primary layer is disabled for that camera, in which case your application must not commit a buffer for that camera.

A buffer created by a [HolographicCamera](#) object can be used only with that object. It should be released when the [HolographicCamera](#) is released, or when the Direct3D 12 device needs to be recreated—whichever happens first. The buffer must not be in the GPU pipeline when it is released—Direct3D 12 fences should be used to ensure that this condition is met prior to releasing the buffer object.

Inheritance

The [IHolographicCameraInterop](#) interface inherits from the [IInspectable](#) interface.

Methods

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

[IHolographicCameraInterop::AcquireDirect3D12BufferResource](#)

The IHolographicCameraInterop::AcquireDirect3D12BufferResource function acquires a Direct3D 12 buffer resource.

[IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout](#)

The IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout function acquires a Direct3D 12 buffer resource, with an optional timeout.

[IHolographicCameraInterop::CreateDirect3D12BackBufferResource](#)

Creates a Direct3D 12 resource for use as a content buffer for the camera.

[IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource](#)

IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource creates a Direct3D 12 resource for use as a content buffer for the camera.

[IHolographicCameraInterop::UnacquireDirect3D12BufferResource](#)

The IHolographicCameraInterop::UnacquireDirect3D12BufferResource function un-acquires a Direct3D 12 buffer resource.

Remarks

To use this interface in [C++/WinRT](#), [QueryInterface](#) for the IHolographicCameraInterop interface from the [HolographicCamera](#) object.

C++/WinRT

```
winrt::com_ptr<IHolographicCameraInterop> spCameraInterop {
    m_holographicCamera.as<IHolographicCameraInterop>() };

D3D12_RESOURCE_DESC bufferDesc { };
bufferDesc.Format =
    SelectFormatUsingHolographicViewConfiguration(
        m_holographicCamera.ViewConfiguration());
bufferDesc.SampleDesc.Count = 1;
bufferDesc.SampleDesc.Quality = 0;
bufferDesc.MipLevels = 1;
bufferDesc.Width = static_cast<UINT64>(
    m_holographicCamera.ViewConfiguration().RenderTargetSize().Width);
bufferDesc.Height = static_cast<UINT64>(
    m_holographicCamera.ViewConfiguration().RenderTargetSize().Height);
```

```

winrt::check_hresult(
    spCameraInterop->CreateDirect3D12BackBufferResource(
        m_deviceResources->GetD3D12Device(),
        &bufferDesc,
        &m_D3D12BackBuffer[m_contentBufferIndex]));

```

You can use the [HolographicViewConfiguration](#) API to determine the available options for buffer format, and to acquire information about render target size for the corresponding output—for example, the [HolographicDisplay](#). If your application needs to change the buffer size for Direct3D 12 buffers from the default render target size for the [HolographicCamera](#), then it should either request a new render target size using the [HolographicViewConfiguration::RequestRenderTargetSize](#) method and create buffers using the size returned by that method, or choose an arbitrary size and override the viewport as described in the following paragraph.

Your Direct3D 12 application can use a viewport size chosen independently by the application. In that case, you must call the [HolographicCameraPose.OverrideViewport](#) method each frame to inform the platform about the viewport used for rendering.

The following code excerpt is from the [Windows Mixed Reality Direct3D 12 app template](#) ↗, which includes boilerplate code for most APIs that are provided in the `Windows.Graphics.Holographic.Interop.h` header.

C++/WinRT

```

winrt::com_ptr<IHolographicCameraInterop> spCameraInterop =
    m_holographicCamera.as<IHolographicCameraInterop>();
winrt::check_hresult(
    spCameraInterop->CreateDirect3D12BackBufferResource(
        spD3D12Device.get(),
        &bufferDesc,
        m_spD3D12BackBuffer[bufferSlot].put()));

```

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraInterop::AcquireDirect3D12BufferResource method (windows.graphics.holographic.interop.h)

Article 08/23/2022

The **AcquireDirect3D12BufferResource** method transitions ownership of a Direct3D 12 back buffer resource from the platform to your application. If your application already owns control of the resource, then the acquisition is still considered to be a success.

After committing a resource to a [HolographicFrame](#) by calling [IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#), your application should consider control of that resource to be owned by the system until such a time as the resource is reacquired by your application using this method. The system owns the buffer until the frame that the buffer was committed to makes its way through the presentation queue. To determine whether the system has relinquished control of the buffer, call **AcquireDirect3D12BufferResource** or **AcquireDirect3D12BufferResourceWithTimeout**. If the buffer can't be acquired by the time your application is ready to start rendering a new [HolographicFrame](#), then you should create a new resource and add it to the buffer queue, or limit the queue size by waiting for a buffer to become available.

If the buffer isn't ready to be acquired when **AcquireDirect3D12BufferResource** is called, then the method call will fail and immediately return the error code **E_NOTREADY**.

Your application can limit the queue size by calling [AcquireDirect3D12BufferResourceWithTimeout](#) to wait until a resource becomes available before queuing more work.

Syntax

C++

```
HRESULT AcquireDirect3D12BufferResource(
    ID3D12Resource     *pResourceToAcquire,
    ID3D12CommandQueue *pCommandQueue
);
```

Parameters

pResourceToAcquire

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to acquire.

pCommandQueue

Type: [ID3D12CommandQueue*](#)

The Direct3D 12 command queue to use for transitioning the state of this resource when acquiring it for your application. The resource will be in the **D3D12_RESOURCE_STATE_COMMON** state when it is acquired. The resource transition command may not be queued if the resource is already in the common state when it is being acquired.

Return value

S_OK if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout method

(windows.graphics.holographic.interop.h)

Article 08/23/2022

The **AcquireDirect3D12BufferResourceWithTimeout** method transitions ownership of a Direct3D 12 back buffer resource from the platform to your application, waiting up to the amount of time indicated by the *duration* argument for the resource to become available. If your application already owns control of the resource, the acquisition is considered to be a success, and the method returns immediately.

After committing a resource to a [HolographicFrame](#) by calling [IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#), your application should consider control of that resource to be owned by the system until such a time as it is reacquired by your application using [AcquireDirect3D12BufferResourceWithTimeout](#). The system owns the buffer until the frame that the buffer was committed to makes its way through the presentation queue. To determine whether the system has relinquished control of the buffer, call [AcquireDirect3D12BufferResource](#) or [AcquireDirect3D12BufferResourceWithTimeout](#). If the buffer can't be acquired by the time your application is ready to start rendering a new [HolographicFrame](#), then you should create a new resource and add it to the buffer queue, or limit the queue size by waiting for a buffer to become available.

This method accepts an optional timeout value. When a nonzero value is present in the *duration* argument, the system waits for that many milliseconds for the buffer to become available. The default behavior is to not wait. When a timeout value of zero is specified and the buffer is not ready to be acquired, the method call fails with the error code **E_NOTREADY**.

Syntax

C++

```
HRESULT AcquireDirect3D12BufferResourceWithTimeout(
```

ID3D12Resource *pResourceToAcquire,	ID3D12CommandQueue *pCommandQueue,
-------------------------------------	------------------------------------

```
    UINT64 duration  
);
```

Parameters

pResourceToAcquire

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to acquire.

pCommandQueue

Type: [ID3D12CommandQueue*](#)

The Direct3D 12 command queue to use for transitioning the state of this resource when acquiring it for your application. The resource will be in the [D3D12_RESOURCE_STATE_COMMON](#) state when it is acquired.

duration

Type: [UINT64](#)

If this parameter is set to a non-zero value, the call will wait for that amount of time for the buffer to be acquired. If the timeout period elapses before the buffer can be acquired, the method will fail with the error code [E_TIMEOUT](#). This parameter is specified in 100-nanosecond units, similar to the [TimeSpan.Duration](#) field.

Return value

[S_OK](#) if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)

Requirement	Value
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraInterop::CreateDirect3D12BackBufferResource method (windows.graphics.holographic.interop.h)

Article 02/22/2024

The **CreateDirect3D12BackBufferResource** method creates a Direct3D 12 resource for use as a back buffer for the corresponding [HolographicCamera](#) API object.

The [D3D12_RESOURCE_DESC](#) structure can contain any set of valid initial values. Any values that won't work with this [HolographicCamera](#) will be overridden in the struct indicated by *pTexture2DDesc*, which is not an optional parameter. The resource is created so that it is already committed to a heap.

Syntax

C++

```
HRESULT CreateDirect3D12BackBufferResource(
    ID3D12Device           *pDevice,
    D3D12_RESOURCE_DESC    *pTexture2DDesc,
    ID3D12Resource         **ppCreatedTexture2DResource
);
```

Parameters

`pDevice`

Type: [ID3D12Device*](#)

A Direct3D 12 device, which will be used to create the resource.

`pTexture2DDesc`

Type: [D3D12_RESOURCE_DESC*](#)

The Direct3D 12 resource description. This parameter is not optional.

CreateDirect3D12BackBufferResource adjusts the description as needed to comply with platform requirements, such as buffer size or format restrictions, which are determined

at runtime. Your application should inspect the descriptor for the texture returned in `ppCreatedTexture2DResource`, and respond appropriately to any differences from what was specified.

`ppCreatedTexture2DResource`

Type: [ID3D12Resource**](#)

If successful, the Direct3D 12 2D texture resource for use as a content buffer. Otherwise, `nullptr`.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[\[+\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource method (windows.graphics.holographic.interop.h)

Article 08/23/2022

The **CreateDirect3D12HardwareProtectedBackBufferResource** method creates a Direct3D 12 resource for use as a back buffer for the corresponding [HolographicCamera](#) API object, with optional hardware-based content protection.

The behavior of **CreateDirect3D12HardwareProtectedBackBufferResource** is the same as that of [CreateDirect3D12BackBufferResource](#), except that it accepts an optional [ID3D12ProtectedResourceSession](#) API object interface pointer. Provide a Direct3D 12 protected resource session via this optional parameter to create a resource buffer with hardware-based content protection enabled.

Syntax

C++

```
HRESULT CreateDirect3D12HardwareProtectedBackBufferResource(
    ID3D12Device                  *pDevice,
    D3D12_RESOURCE_DESC             *pTexture2DDesc,
    ID3D12ProtectedResourceSession *pProtectedResourceSession,
    ID3D12Resource                 **ppCreatedTexture2DResource
);
```

Parameters

pDevice

Type: [ID3D12Device*](#)

A Direct3D 12 device, which will be used to create the resource.

pTexture2DDesc

Type: [D3D12_RESOURCE_DESC*](#)

The Direct3D 12 resource description.

`CreateDirect3D12HardwareProtectedBackBufferResource` adjusts the description as needed to comply with platform requirements, such as buffer size or format restrictions, which are determined at runtime. Your application should inspect the descriptor for the texture returned in `ppCreatedTexture2DResource` and respond appropriately to any differences from what was specified.

`pProtectedResourceSession`

Type: [ID3D12ProtectedResourceSession*](#)

An optional Direct3D 12 protected resource session. Passing in a valid protected session will cause this method to create a Direct3D 12 hardware-protected resource.

`ppCreatedTexture2DResource`

Type: [ID3D12Resource**](#)

If successful, the hardware-protected Direct3D 12 2D texture resource for use as a back buffer. Otherwise, `nullptr`.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

IHolographicCameraInterop::UnacquireDirect3D12BufferResource method (windows.graphics.holographic.interop.h)

Article 02/22/2024

The **UnacquireDirect3D12BufferResource** method relinquishes control of a Direct3D 12 buffer resource to the platform.

A resource that has been acquired, but not submitted, can be un-acquired to return control of the buffer back to the platform. A resource that has been un-acquired can be re-acquired at a later time.

Syntax

C++

```
HRESULT UnacquireDirect3D12BufferResource(  
    ID3D12Resource *pResourceToUnacquire  
>;
```

Parameters

`pResourceToUnacquire`

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to relinquish control of.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraRenderingParametersInterop interface (windows.graphics.holographic.interop.h)

Article 08/23/2022

The [IHolographicCameraRenderingParametersInterop](#) interface is a nano-COM interface, used to commit Direct3D 12 buffer resources for presentation during the corresponding [HolographicFrame](#).

The interface allows COM interop with the [HolographicCameraRenderingParameters](#) Windows Runtime class for applications that use Direct3D 12 for holographic rendering. Nano-COM allows Direct3D 12 objects to be used directly as parameters for API calls, rather than going through a container object.

Inheritance

The [IHolographicCameraRenderingParametersInterop](#) interface inherits from the [IInspectable](#) interface.

Methods

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

Expand table

[IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource](#)

The [IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource](#) function commits a Direct3D 12 buffer for presentation on HolographicCamera outputs.

[IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData](#)

The [IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData](#) function commits a Direct3D 12 buffer for HolographicCamera outputs.

Remarks

To use this interface in [C++/WinRT](#), retrieve the [HolographicCameraRenderingParameters](#) object from the [HolographicFrame](#), and then [QueryInterface](#) for the [IHolographicCameraRenderingParametersInterop](#) interface.

C++/WinRT

```
auto holographicCameraRenderingParameters {  
    holographicFrame.GetRenderingParameters(m_cameraPose) };  
winrt::com_ptr<IHolographicCameraRenderingParametersInterop>  
holographicCameraRenderingParametersInterop  
{  
  
    holographicCameraRenderingParameters.as<IHolographicCameraRenderingParameter  
sInterop>();  
};
```

To use this interface in C++/CX, first cast the [HolographicCameraRenderingParameters](#) object (after retrieving it from the [HolographicFrame](#)) to [IInspectable](#)*. Then [QueryInterface](#) for the [IHolographicCameraRenderingParametersInterop](#) interface from the [IInspectable](#) pointer.

C++/CX

```
auto holographicCameraRenderingParameters =  
    holographicFrame->GetRenderingParameters(m_cameraPose);  
Microsoft::WRL::ComPtr<IHolographicCameraRenderingParametersInterop>  
holographicCameraRenderingParametersInterop;  
{  
    Microsoft::WRL::ComPtr<IInspectable> iInspectable =  
    reinterpret_cast<IInspectable*>(holographicCameraRenderingParameters);  
  
    DX::ThrowIfFailed(iInspectable.As(&holographicCameraRenderingParametersInter  
op));  
}
```

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)

Requirement	Value
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource method (windows.graphics.holographic.interop.h)

Article 08/23/2022

The [CommitDirect3D12Resource](#) method commits a Direct3D 12 buffer for presentation on outputs associated with a [HolographicCamera](#) during a specific [HolographicFrame](#). The buffer must have been created by calling [CreateDirect3D12BackBufferResource](#) or [CreateDirect3D12HardwareProtectedBackBufferResource](#) on the same [HolographicCamera](#) corresponding to this rendering parameters object, and the buffer must have been acquired by your application prior to rendering.

Syntax

C++

```
HRESULT CommitDirect3D12Resource(
    ID3D12Resource *pColorResourceToCommit,
    ID3D12Fence     *pColorResourceFence,
    UINT64           colorResourceFenceSignalValue
);
```

Parameters

pColorResourceToCommit

Type: [ID3D12Resource*](#)

The Direct3D 12 texture resource with content to display when presenting the [HolographicFrame](#) used to retrieve this rendering parameters object.

pColorResourceFence

Type: [ID3D12Fence*](#)

A fence used to signal app work completion on the color buffer resource indicated by *pColorResourceToCommit*. Completion of this fence at the value indicated by *colorResourceFenceSignalValue* signals transfer of control of the color resource from your application to the platform in the GPU work queue. The platform relies upon this fence, and the value indicated in *colorResourceFenceSignalValue*, to queue work on the GPU that reads from the color buffer.

`colorResourceFenceSignalValue`

Type: [UINT64](#)

The value used to signal work completion on *pColorResourceFence*. The platform relies upon this fence value to queue work on the GPU that reads from the color buffer.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData method (windows.graphics.holographic.interop.h)

Article 08/23/2022

Commits a Direct3D 12 buffer for presentation on outputs associated with the [HolographicCamera](#). The buffer must have been created by calling [CreateDirect3D12BackBufferResource](#) or [CreateDirect3D12HardwareProtectedBackBufferResource](#) on the same [HolographicCamera](#) that it is committed for.

This method also accepts an optional depth buffer parameter, along with fence and fence value for app work completion on that buffer. This depth buffer will be used for image stabilization when showing the frame it is committed to. The depth buffer must contain depth data correlated with geometry used to draw holograms in the color buffer, which is submitted at the same time. The depth buffer should not contain depth data for invisible content, such as depth data used for occlusion.

Syntax

C++

```
HRESULT CommitDirect3D12ResourceWithDepthData(
    ID3D12Resource *pColorResourceToCommit,
    ID3D12Fence     *pColorResourceFence,
    UINT64          colorResourceFenceSignalValue,
    ID3D12Resource *pDepthResourceToCommit,
    ID3D12Fence     *pDepthResourceFence,
    UINT64          depthResourceFenceSignalValue
);
```

Parameters

pColorResourceToCommit

Type: [ID3D12Resource*](#)

The Direct3D 12 texture resource with content to display when presenting the [HolographicFrame](#) used to retrieve this rendering parameters object.

`pColorResourceFence`

Type: [ID3D12Fence*](#)

A fence used to signal app work completion on the color buffer resource indicated by *pColorResourceToCommit*. Completion of this fence at the value indicated by *colorResourceFenceSignalValue* signals transfer of control of the color resource from your application to the platform in the GPU work queue. The platform relies upon this fence, and the value indicated in *colorResourceFenceSignalValue*, to queue work on the GPU that reads from the color buffer.

`colorResourceFenceSignalValue`

Type: [UINT64](#)

The value used to signal work completion on *pColorResourceFence*. The platform relies upon this fence value to queue work on the GPU that reads from the color buffer.

`pDepthResourceToCommit`

Type: [ID3D12Resource*](#)

The Direct3D 12 depth buffer with depth data to use for image stabilization when presenting the [HolographicFrame](#) used to retrieve this rendering parameters object. Applications typically submit the depth stencil used when rendering to *pColorResourceToCommit*, or a depth buffer that is derived from the same rendering pass. The depth buffer should only include data corresponding to geometry used to render holograms in the color buffer; for example, occlusion data shouldn't be included, and may be ignored by the platform.

`pDepthResourceFence`

Type: [ID3D12Fence*](#)

A fence used to signal work completion on the depth buffer resource indicated by *pDepthResourceToCommit*. Completion of this fence at the value indicated by *depthResourceFenceSignalValue* signals transfer of control of the depth resource from your application to the platform in the GPU work queue. The platform relies upon this fence, and the value indicated in *colorResourceFenceSignalValue*, to queue work on the GPU that reads from the depth buffer.

`depthResourceFenceSignalValue`

Type: [UINT64](#)

The value used to signal work completion on *pDepthResourceFence*. The platform relies upon this fence value to queue work on the GPU that reads from the depth buffer.

Return value

[S_OK](#) if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

 [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IHolographicQuadLayerInterop interface (windows.graphics.holographic.interop.h)

Article08/23/2022

The **IHolographicQuadLayerInterop** interface is a nano-COM interface, used to create Direct3D 12 content buffers for a [HolographicQuadLayer](#) Windows Runtime object. This is an initialization step for using Direct3D 12 with Windows Mixed Reality quad layers. It also allows your application to acquire ownership of content buffers for rendering, prior to committing them with the [IHolographicQuadLayerUpdateParametersInterop](#) interface.

Your application can use **IHolographicQuadLayerInterop** to initialize Direct3D 12 content buffer resources for holographic quad layers. Nano-COM allows pointers to Direct3D 12 objects to be passed directly as parameters for API calls, instead of using a Windows Runtime container object.

Your application manages its own pool of holographic content buffer resources. It can create additional buffers as needed in order to continue rendering smoothly. On most devices, this will be three or four buffers. Your application should start with at least two buffers in the pool. Your application can dynamically detect when it needs to create a new buffer by looking for failed attempts to immediately acquire buffers that were previously committed for presentation. A quad layer content buffer will continue to be presented each frame until a new buffer is committed.

A buffer created by a [HolographicQuadLayer](#) object can be used only with that object. It should be released when the [HolographicQuadLayer](#) is released, or when the Direct3D 12 device needs to be recreated—whichever happens first. The buffer must not be in the GPU pipeline when it is released—Direct3D 12 fences should be used to ensure that this condition is met prior to releasing the buffer object.

Inheritance

The **IHolographicQuadLayerInterop** interface inherits from the [IInspectable](#) interface.

Methods

The **IHolographicQuadLayerInterop** interface has these methods.

[IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource](#)

The IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource function acquires a Direct3D 12 buffer resource.

[IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout](#)

The IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout function acquires a Direct3D 12 buffer resource, with an optional timeout.

[IHolographicQuadLayerInterop::CreateDirect3D12ContentBufferResource](#)

Creates a Direct3D 12 resource for use as a content buffer for the layer.

[IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource](#)

The IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource function creates a Direct3D 12 resource content buffer for the camera.

[IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource](#)

The IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource function un-acquires a Direct3D 12 buffer resource.

Remarks

To use this interface in [C++/WinRT](#), QueryInterface for the **IHolographicQuadLayerInterop** interface from the [HolographicQuadLayer](#) object.

Note that you can use the [HolographicViewConfiguration](#) API to determine the available options for buffer format.

C++/WinRT

```
m_quadLayer = HolographicQuadLayer{ {1024, 1024} };
winrt::com_ptr<IHolographicQuadLayerInterop> quadLayerInterop{
    m_quadLayer.as<IHolographicQuadLayerInterop>() };

// Create/acquire buffer.
if (!m_D3D12ContentBuffer[m_contentBufferIndex])
{
    D3D12_RESOURCE_DESC bufferDesc{ sourceDesc };
    bufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
    bufferDesc.SampleDesc.Count = 1;
    bufferDesc.SampleDesc.Quality = 0;
```

```

        bufferDesc.MipLevels = 1;

        winrt::check_hresult(
            quadLayerInterop->CreateDirect3D12ContentBufferResource(
                m_deviceResources->GetD3D12Device(),
                &bufferDesc,
                &m_D3D12ContentBuffer[m_contentBufferIndex]));
    }
}

```

To use this interface in C++/CX, cast the [HolographicQuadLayer](#) object to [IInspectable](#)*. Then QueryInterface for the [IHolographicQuadLayerInterop](#) interface from the [IInspectable](#) pointer.

C++/CX

```

m_quadLayer = ref new HolographicQuadLayer();
Microsoft::WRL::ComPtr<IHolographicQuadLayerInterop> quadLayerInterop;
{
    Microsoft::WRL::ComPtr<IIInspectable> iInspectable =
    reinterpret_cast<IIInspectable*>(m_quadLayer);
    DX::ThrowIfFailed(iInspectable.As(&quadLayerInterop));
}

// Create/acquire buffer.
if (!m_D3D12ContentBuffer[m_contentBufferIndex])
{
    D3D12_RESOURCE_DESC bufferDesc = sourceDesc;
    bufferDesc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
    bufferDesc.SampleDesc.Count = 1;
    bufferDesc.SampleDesc.Quality = 0;
    bufferDesc.MipLevels = 1;

    DX::ThrowIfFailed(quadLayerInterop-
        >CreateDirect3D12ContentBufferResource(
            m_deviceResources->GetD3D12Device(),
            &bufferDesc,
            &m_D3D12ContentBuffer[m_contentBufferIndex]));
}

```

Requirements

[\[\] Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)

Requirement	Value
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource method (windows.graphics.holographic.interop.h)

Article08/23/2022

The **AcquireDirect3D12BufferResource** method transitions ownership of a Direct3D 12 content buffer resource from the platform to your application. If your application already owns control of the resource, then the acquisition is still considered to be a success.

After committing a resource to a [HolographicFrame](#) by calling [IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#), your application should consider control of that resource to be relinquished to the system until such a time as it is reacquired by your application using **AcquireDirect3D12BufferResource**. The system owns the buffer until it is no longer needed for presenting the quad layer. To determine whether the system has relinquished control of the buffer, call **AcquireDirect3D12BufferResource** or **AcquireDirect3D12BufferResourceWithTimeout**. If the buffer can't be acquired by the time your application is ready to start rendering a new update for the quad layer, then you should create a new resource and add it to the buffer queue, or limit the queue size by waiting for a buffer to become available.

If the buffer is not ready to be acquired when this method is called, the method call fails and immediately returns the error code **E_NOTREADY**.

Your application can limit the queue size by calling [AcquireDirect3D12BufferResourceWithTimeout](#) to wait until a resource becomes available before queuing more work.

Syntax

C++

```
HRESULT AcquireDirect3D12BufferResource(
    ID3D12Resource     *pResourceToAcquire,
    ID3D12CommandQueue *pCommandQueue
);
```

Parameters

pResourceToAcquire

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to acquire. The resource will be in the **D3D12_RESOURCE_STATE_COMMON** state when it is acquired.

pCommandQueue

Type: [ID3D12CommandQueue*](#)

The Direct3D 12 command queue to use for transitioning the state of this resource when acquiring it for your application.

Return value

S_OK if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

See also

[AcquireDirect3D12BufferResourceWithTimeout](#)

Feedback

Was this page helpful?

 Yes

 No

IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout method (windows.graphics.holographic.interop.h)

Article 08/23/2022

The **AcquireDirect3D12BufferResourceWithTimeout** method transitions ownership of a Direct3D 12 back buffer resource from the platform to your application, waiting up to the amount of time indicated by the *duration* argument for the resource to become available. If your application already owns control of the resource, then the acquisition is still considered to be a success, and the method returns immediately.

After committing a resource to a [HolographicFrame](#) by calling [IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#), your application should consider control of that resource to be relinquished to the system until such a time as it is reacquired by your application using [AcquireDirect3D12BufferResourceWithTimeout](#). The system owns the buffer until it is no longer needed for presenting the quad layer. To determine whether the system has relinquished control of the buffer, call [AcquireDirect3D12BufferResource](#) or [AcquireDirect3D12BufferResourceWithTimeout](#). If the buffer can't be acquired by the time your application is ready to start rendering a new update for the quad layer, then you should create a new resource and add it to the buffer queue, or limit the queue size by waiting for a buffer to become available.

This method accepts an optional timeout value. When a non-zero value is present in the *duration* argument, the system waits for that many milliseconds for the buffer to become available. The default behavior is to not wait. When a timeout value of zero is specified, and the buffer is not ready to be acquired, the method call fails with the error code [E_NOTREADY](#).

Syntax

C++

```
HRESULT AcquireDirect3D12BufferResourceWithTimeout(  
    ID3D12Resource     *pResourceToAcquire,  
    ID3D12CommandQueue *pCommandQueue,
```

```
    UINT64 duration  
);
```

Parameters

pResourceToAcquire

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to acquire. The resource will be in the **D3D12_RESOURCE_STATE_COMMON** state when it is acquired.

pCommandQueue

Type: [ID3D12CommandQueue*](#)

The Direct3D 12 command queue to use for transitioning the state of this resource when acquiring it for your application.

duration

Type: [UINT64](#)

If this parameter is set, the call will wait for that amount of time for the buffer to be acquired. If the timeout period elapses before the buffer can be acquired, the method fails with the error code **E_TIMEOUT**. This parameter is in 100-nanosecond units, similar to the [TimeSpan.Duration](#) field.

Return value

S_OK if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

When no timeout value is specified, if this method is called and the buffer is not ready to be acquired, the method call will fail with the error code **E_NOTREADY**.

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)

Requirement	Value
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

See also

[AcquireDirect3D12BufferResource](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

IHolographicQuadLayerInterop::CreateDirect3D12ContentBufferResource method (windows.graphics.holographic.interop.h)

Article02/22/2024

The `CreateDirect3D12ContentBufferResource` method creates a Direct3D 12 resource for use as a back buffer for the corresponding [HolographicQuadLayer](#) API object.

The [`D3D12_RESOURCE_DESC`](#) structure can contain any set of valid initial values. Any values that won't work with this quad layer object will be overridden in the struct indicated by `pTexture2DDesc`, which is not an optional parameter. The resource is created so that it is already committed to a heap.

Syntax

C++

```
HRESULT CreateDirect3D12ContentBufferResource(
    ID3D12Device             *pDevice,
    D3D12_RESOURCE_DESC       *pTexture2DDesc,
    ID3D12Resource           **ppTexture2DResource
);
```

Parameters

`pDevice`

Type: [`ID3D12Device`*](#)

A Direct3D 12 device, which will be used to create the resource.

`pTexture2DDesc`

Type: [`D3D12_RESOURCE_DESC`*](#)

The Direct3D 12 resource description. This parameter is not optional.

`CreateDirect3D12ContentBufferResource` adjusts the description as needed to comply with platform requirements, such as buffer size or format restrictions, which are determined at runtime. Your application should inspect the descriptor for the texture returned in `ppCreatedTexture2DResource`, and respond appropriately to any differences from what was specified.

`ppTexture2DResource`

Type: [ID3D12Resource**](#)

If successful, the Direct3D 12 2D texture resource for use as a content buffer. Otherwise, `nullptr`.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource method (windows.graphics.holographic.interop.h)

Article02/22/2024

The **CreateDirect3D12HardwareProtectedContentBufferResource** method creates a Direct3D 12 resource for use as a back buffer for the corresponding [HolographicQuadLayer](#) API object, with optional hardware-based content protection.

The behavior of **CreateDirect3D12HardwareProtectedContentBufferResource** is the same as that of [CreateDirect3D12ContentBufferResource](#), except that it accepts an optional [ID3D12ProtectedResourceSession](#) API object interface pointer. Provide a Direct3D 12 protected resource session via this optional parameter to create a resource buffer with hardware-based content protection enabled.

Syntax

C++

```
HRESULT CreateDirect3D12HardwareProtectedContentBufferResource(
    ID3D12Device                  *pDevice,
    D3D12_RESOURCE_DESC             *pTexture2DDesc,
    ID3D12ProtectedResourceSession *pProtectedResourceSession,
    ID3D12Resource                 **ppCreatedTexture2DResource
);
```

Parameters

pDevice

Type: [ID3D12Device*](#)

A Direct3D 12 device, which will be used to create the resource.

pTexture2DDesc

Type: [D3D12_RESOURCE_DESC*](#)

The Direct3D 12 resource description.

`CreateDirect3D12HardwareProtectedContentBufferResource` adjusts the description as needed to comply with platform requirements, such as buffer size or format restrictions, which are determined at runtime. Your application should inspect the descriptor for the texture returned in `ppCreatedTexture2DResource` and respond appropriately to any differences from what was specified.

`pProtectedResourceSession`

Type: [ID3D12ProtectedResourceSession*](#)

An optional Direct3D 12 protected resource session. Passing in a valid protected session causes this method to create a Direct3D 12 hardware-protected resource.

`ppCreatedTexture2DResource`

Type: [ID3D12Resource**](#)

If successful, the hardware-protected Direct3D 12 2D texture resource for use as a content buffer. Otherwise, `nullptr`.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the error code reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

Provide product feedback  | Get help at Microsoft Q&A

IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource method (windows.graphics.holographic.interop.h)

Article02/22/2024

The **UnacquireDirect3D12BufferResource** method relinquishes control of a Direct3D 12 buffer resource to the platform.

A resource that has been acquired, but not submitted, can be un-acquired to return control of the buffer back to the platform. A resource that has been un-acquired can be re-acquired at a later time.

Syntax

C++

```
HRESULT UnacquireDirect3D12BufferResource(  
    ID3D12Resource *pResourceToUnacquire  
>;
```

Parameters

`pResourceToUnacquire`

Type: [ID3D12Resource*](#)

The Direct3D 12 resource to relinquish control of.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

IHolographicQuadLayerUpdateParametersInterop interface (windows.graphics.holographic.interop.h)

Article 02/22/2024

The **IHolographicQuadLayerUpdateParametersInterop** interface is a nano-COM interface, used to commit Direct3D 12 buffer resources for quad layer rendering in the corresponding [HolographicFrame](#).

The interface allows COM interop with the [HolographicQuadLayerUpdateParameters](#) class for applications that use Direct3D 12 for holographic rendering. Nano-COM allows Direct3D 12 objects to be used directly as parameters for API calls, rather than going through a container object.

Inheritance

The **IHolographicQuadLayerUpdateParametersInterop** interface inherits from the [IInspectable](#) interface.

Methods

The **IHolographicQuadLayerUpdateParametersInterop** interface has these methods.

[] [Expand table](#)

[IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#)

Commits a Direct3D 12 buffer for presentation on outputs associated with any [HolographicCamera](#) to which the quad layer is attached.

Remarks

To use this interface in [C++/WinRT](#), retrieve the [HolographicQuadLayerUpdateParameters](#) object from the [HolographicFrame](#), and then QueryInterface for the **IHolographicQuadLayerUpdateParametersInterop** interface.

```
auto quadLayerParameters{
    holographicFrame.GetQuadLayerUpdateParameters(m_quadLayer) };
winrt::com_ptr<IHolographicQuadLayerUpdateParametersInterop>
quadLayerParametersInterop{
    quadLayerParameters.as<IHolographicQuadLayerUpdateParametersInterop>()
};
```

To use this interface in C++/CX, first cast the `HolographicQuadLayerUpdateParameters` object (after retrieving it from the `HolographicFrame`) to `IInspectable`*. Then QueryInterface for the `IHolographicQuadLayerUpdateParametersInterop` interface from the `IInspectable` pointer.

```
auto quadLayerParameters = holographicFrame-
>GetQuadLayerUpdateParameters(m_quadLayer);
Microsoft::WRL::ComPtr<IHolographicQuadLayerUpdateParametersInterop>
quadLayerParametersInterop;
{
    Microsoft::WRL::ComPtr<IInspectable> iInspectable =
reinterpret_cast<IInspectable*>(quadLayerParameters);
    DX::ThrowIfFailed(iInspectable.As(&quadLayerParamsInterop));
}
```

Requirements

[] Expand table

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

Yes

No

IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource method (windows.graphics.holographic.Interop.h)

Article 02/22/2024

The `CommitDirect3D12Resource` method commits a Direct3D 12 buffer for presentation on outputs associated with any [HolographicCamera](#) to which the quad layer is attached. The buffer must have been created by calling [CreateDirect3D12ContentBufferResource](#) or [CreateDirect3D12HardwareProtectedContentBufferResource](#) on the same [HolographicQuadLayer](#) corresponding to this update parameters object, and the buffer must have been acquired by your application prior to rendering.

Syntax

C++

```
HRESULT CommitDirect3D12Resource(
    ID3D12Resource *pColorResourceToCommit,
    ID3D12Fence     *pColorResourceFence,
    UINT64           colorResourceFenceSignalValue
);
```

Parameters

pColorResourceToCommit

Type: [ID3D12Resource](#)*

The Direct3D 12 texture resource with content to display when rendering the [HolographicQuadLayer](#) corresponding to this update parameters object. The content will also be displayed during any subsequent frames, until another content buffer update is provided for this [HolographicQuadLayer](#).

pColorResourceFence

Type: [ID3D12Fence](#)*

A fence used to signal app work completion on the content buffer resource indicated by *pColorResourceToCommit*. Completion of this fence at the value indicated by *colorResourceFenceSignalValue* signals transfer of control of the content buffer resource from your application to the platform in the GPU work queue. The platform relies upon this fence, and the value indicated in *colorResourceFenceSignalValue*, to queue work on the GPU that reads from the content buffer.

`colorResourceFenceSignalValue`

Type: [UINT64](#)

The value used to signal work completion on *pColorResourceFence*. The platform relies upon this fence value to queue work on the GPU that reads from the content buffer.

Return value

`S_OK` if successful, otherwise returns an [HRESULT](#) error code indicating the reason for failure. Also see [COM Error Codes \(UI, Audio, DirectX, Codec\)](#).

Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	windows.graphics.holographic.interop.h

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)