

Contents

Direct3D 12 Graphics

D3d11on12.h

Overview

D3D11_RESOURCE_FLAGS structure

D3D11On12CreateDevice function

ID3D11On12Device interface

 Overview

 ID3D11On12Device::AcquireWrappedResources method

 ID3D11On12Device::CreateWrappedResource method

 ID3D11On12Device::ReleaseWrappedResources method

D3d12.h

Overview

D3D_ROOT_SIGNATURE_VERSION enumeration

D3D_SHADER_MODEL enumeration

D3D12_AUTO_BREADCRUMB_NODE structure

D3D12_BACKGROUND_PROCESSING_MODE enumeration

D3D12_BLEND enumeration

D3D12_BLEND_DESC structure

D3D12_BLEND_OP enumeration

D3D12_BOX structure

D3D12_BUFFER_RTV structure

D3D12_BUFFER_SRV structure

D3D12_BUFFER_SRV_FLAGS enumeration

D3D12_BUFFER_UAV structure

D3D12_BUFFER_UAV_FLAGS enumeration

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC structure

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS structure

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER structure

D3D12_CACHED_PIPELINE_STATE structure

D3D12_CLEAR_FLAGS enumeration

D3D12_CLEAR_VALUE structure

D3D12_COLOR_WRITE_ENABLE enumeration

D3D12_COMMAND_LIST_SUPPORT_FLAGS enumeration

D3D12_COMMAND_LIST_TYPE enumeration

D3D12_COMMAND_QUEUE_DESC structure

D3D12_COMMAND_QUEUE_FLAGS enumeration
D3D12_COMMAND_QUEUE_PRIORITY enumeration
D3D12_COMMAND_SIGNATURE_DESC structure
D3D12_COMPARISON_FUNC enumeration
D3D12_COMPUTE_PIPELINE_STATE_DESC structure
D3D12_CONSERVATIVE_RASTERIZATION_MODE enumeration
D3D12_CONSERVATIVE_RASTERIZATION_TIER enumeration
D3D12_CONSTANT_BUFFER_VIEW_DESC structure
D3D12_CPU_DESCRIPTOR_HANDLE structure
D3D12_CPU_PAGE_PROPERTY enumeration
D3D12_CROSS_NODE_SHARING_TIER enumeration
D3D12_CULL_MODE enumeration
D3D12_DEPTH_STENCIL_DESC structure
D3D12_DEPTH_STENCIL_DESC1 structure
D3D12_DEPTH_STENCIL_VALUE structure
D3D12_DEPTH_STENCIL_VIEW_DESC structure
D3D12_DEPTH_STENCILOP_DESC structure
D3D12_DEPTH_WRITE_MASK enumeration
D3D12_DESCRIPTOR_HEAP_DESC structure
D3D12_DESCRIPTOR_HEAP_FLAGS enumeration
D3D12_DESCRIPTOR_HEAP_TYPE enumeration
D3D12_DESCRIPTOR_RANGE structure
D3D12_DESCRIPTOR_RANGE_FLAGS enumeration
D3D12_DESCRIPTOR_RANGE_TYPE enumeration
D3D12_DESCRIPTOR_RANGE1 structure
D3D12_DEVICE_REMOVED_EXTENDED_DATA structure
D3D12_DEVICE_REMOVED_EXTENDED_DATA1 structure
D3D12_DISCARD_REGION structure
D3D12_DISPATCH_ARGUMENTS structure
D3D12_DISPATCH_RAYS_DESC structure
D3D12_DRAW_ARGUMENTS structure
D3D12_DRAW_INDEXED_ARGUMENTS structure
D3D12_DRED_ALLOCATION_NODE structure
D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT structure
D3D12_DRED_PAGE_FAULT_OUTPUT structure
D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS enumeration
D3D12_DSV_DIMENSION enumeration
D3D12_DSV_FLAGS enumeration
D3D12_DXIL_LIBRARY_DESC structure

D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure
D3D12_ELEMENTS_LAYOUT enumeration
D3D12_EXISTING_COLLECTION_DESC structure
D3D12_EXPORT_DESC structure
D3D12_EXPORT_FLAGS enumeration
D3D12_FEATURE enumeration
D3D12_FEATURE_DATA_ARCHITECTURE structure
D3D12_FEATURE_DATA_ARCHITECTURE1 structure
D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY structure
D3D12_FEATURE_DATA_D3D12_OPTIONS structure
D3D12_FEATURE_DATA_D3D12_OPTIONS1 structure
D3D12_FEATURE_DATA_D3D12_OPTIONS2 structure
D3D12_FEATURE_DATA_D3D12_OPTIONS3 structure
D3D12_FEATURE_DATA_D3D12_OPTIONS4 structure
D3D12_FEATURE_DATA_D3D12_OPTIONS5 structure
D3D12_FEATURE_DATA_D3D12_OPTIONS6 structure
D3D12_FEATURE_DATA_EXISTING_HEAPS structure
D3D12_FEATURE_DATA_FEATURE_LEVELS structure
D3D12_FEATURE_DATA_FORMAT_INFO structure
D3D12_FEATURE_DATA_FORMAT_SUPPORT structure
D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT structure
D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS structure
D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT structure
D3D12_FEATURE_DATA_QUERY_META_COMMAND structure
D3D12_FEATURE_DATA_ROOT_SIGNATURE structure
D3D12_FEATURE_DATA_SERIALIZATION structure
D3D12_FEATURE_DATA_SHADER_CACHE structure
D3D12_FEATURE_DATA_SHADER_MODEL structure
D3D12_FENCE_FLAGS enumeration
D3D12_FILL_MODE enumeration
D3D12_FILTER enumeration
D3D12_FILTER_REDUCTION_TYPE enumeration
D3D12_FILTER_TYPE enumeration
D3D12_FORMAT_SUPPORT1 enumeration
D3D12_FORMAT_SUPPORT2 enumeration
D3D12_GLOBAL_ROOT_SIGNATURE structure
D3D12_GPU_DESCRIPTOR_HANDLE structure
D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE structure
D3D12_GPU_VIRTUAL_ADDRESS_RANGE structure

D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE structure
D3D12_GRAPHICS_PIPELINE_STATE_DESC structure
D3D12_GRAPHICS_STATES enumeration
D3D12_HEAP_DESC structure
D3D12_HEAP_FLAGS enumeration
D3D12_HEAP_PROPERTIES structure
D3D12_HEAP_TYPE enumeration
D3D12_HIT_GROUP_DESC structure
D3D12_HIT_GROUP_TYPE enumeration
D3D12_INDEX_BUFFER_STRIP_CUT_VALUE enumeration
D3D12_INDEX_BUFFER_VIEW structure
D3D12_INDIRECT_ARGUMENT_DESC structure
D3D12_INDIRECT_ARGUMENT_TYPE enumeration
D3D12_INPUT_CLASSIFICATION enumeration
D3D12_INPUT_ELEMENT_DESC structure
D3D12_INPUT_LAYOUT_DESC structure
D3D12_LOCAL_ROOT_SIGNATURE structure
D3D12_LOGIC_OP enumeration
D3D12_MEASUREMENTS_ACTION enumeration
D3D12_MEMCPY_DEST structure
D3D12_MEMORY_POOL enumeration
D3D12_META_COMMAND_DESC structure
D3D12_META_COMMAND_PARAMETER_DESC structure
D3D12_META_COMMAND_PARAMETER_FLAGS enumeration
D3D12_META_COMMAND_PARAMETER_STAGE enumeration
D3D12_META_COMMAND_PARAMETER_TYPE enumeration
D3D12_MULTIPLE_FENCE_WAIT_FLAGS enumeration
D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS enumeration
D3D12_NODE_MASK structure
D3D12_PACKED_MIP_INFO structure
D3D12_PIPELINE_STATE_FLAGS enumeration
D3D12_PIPELINE_STATE_STREAM_DESC structure
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE enumeration
D3D12_PLACED_SUBRESOURCE_FOOTPRINT structure
D3D12_PREDICATION_OP enumeration
D3D12_PRIMITIVE_TOPOLOGY_TYPE enumeration
D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER enumeration
D3D12_PROTECTED_RESOURCE_SESSION_DESC structure
D3D12_PROTECTED_RESOURCE_SESSION_FLAGS enumeration

D3D12_QUERY_DATA_PIPELINE_STATISTICS structure
D3D12_QUERY_DATA_SO_STATISTICS structure
D3D12_QUERY_HEAP_DESC structure
D3D12_QUERY_HEAP_TYPE enumeration
D3D12_QUERY_TYPE enumeration
D3D12_RANGE structure
D3D12_RANGE_UINT64 structure
D3D12_RASTERIZER_DESC structure
D3D12_RAY_FLAGS enumeration
D3D12_RAYTRACING_AABB structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS enumeration
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE enumeration
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC structure

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC structure

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE enumeration
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV structure
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE enumeration
D3D12_RAYTRACING_GEOMETRY_AABBS_DESC structure
D3D12_RAYTRACING_GEOMETRY_DESC structure
D3D12_RAYTRACING_GEOMETRY_FLAGS enumeration
D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC structure
D3D12_RAYTRACING_GEOMETRY_TYPE enumeration
D3D12_RAYTRACING_INSTANCE_DESC structure
D3D12_RAYTRACING_INSTANCE_FLAGS enumeration
D3D12_RAYTRACING_PIPELINE_CONFIG structure
D3D12_RAYTRACING_SHADER_CONFIG structure
D3D12_RAYTRACING_TIER enumeration
D3D12_RENDER_PASS_BEGINNING_ACCESS structure
D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS structure
D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE enumeration
D3D12_RENDER_PASS_DEPTH_STENCIL_DESC structure
D3D12_RENDER_PASS_ENDING_ACCESS structure

D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS structure
D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS structure
D3D12_RENDER_PASS_ENDING_ACCESS_TYPE enumeration
D3D12_RENDER_PASS_FLAGS enumeration
D3D12_RENDER_PASS_RENDER_TARGET_DESC structure
D3D12_RENDER_PASS_TIER enumeration
D3D12_RENDER_TARGET_BLEND_DESC structure
D3D12_RENDER_TARGET_VIEW_DESC structure
D3D12_RESIDENCY_FLAGS enumeration
D3D12_RESIDENCY_PRIORITY enumeration
D3D12_RESOLVE_MODE enumeration
D3D12_RESOURCE_ALIASING_BARRIER structure
D3D12_RESOURCE_ALLOCATION_INFO structure
D3D12_RESOURCE_ALLOCATION_INFO1 structure
D3D12_RESOURCE_BARRIER structure
D3D12_RESOURCE_BARRIER_FLAGS enumeration
D3D12_RESOURCE_BARRIER_TYPE enumeration
D3D12_RESOURCE_BINDING_TIER enumeration
D3D12_RESOURCE_DESC structure
D3D12_RESOURCE_DIMENSION enumeration
D3D12_RESOURCE_FLAGS enumeration
D3D12_RESOURCE_HEAP_TIER enumeration
D3D12_RESOURCE_STATES enumeration
D3D12_RESOURCE_TRANSITION_BARRIER structure
D3D12_RESOURCE_UAV_BARRIER structure
D3D12_ROOT_CONSTANTS structure
D3D12_ROOT_DESCRIPTOR structure
D3D12_ROOT_DESCRIPTOR_FLAGS enumeration
D3D12_ROOT_DESCRIPTOR_TABLE structure
D3D12_ROOT_DESCRIPTOR_TABLE1 structure
D3D12_ROOT_DESCRIPTOR1 structure
D3D12_ROOT_PARAMETER structure
D3D12_ROOT_PARAMETER_TYPE enumeration
D3D12_ROOT_PARAMETER1 structure
D3D12_ROOT_SIGNATURE_DESC structure
D3D12_ROOT_SIGNATURE_DESC1 structure
D3D12_ROOT_SIGNATURE_FLAGS enumeration
D3D12_RT_FORMAT_ARRAY structure
D3D12_RTV_DIMENSION enumeration

D3D12_SAMPLE_POSITION structure
D3D12_SAMPLER_DESC structure
D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER structure
D3D12_SERIALIZED_DATA_TYPE enumeration
D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER structure
D3D12_SHADER_BYTECODE structure
D3D12_SHADER_CACHE_SUPPORT_FLAGS enumeration
D3D12_SHADER_COMPONENT_MAPPING enumeration
D3D12_SHADER_MIN_PRECISION_SUPPORT enumeration
D3D12_SHADER_RESOURCE_VIEW_DESC structure
D3D12_SHADER_VISIBILITY enumeration
D3D12_SO_DECLARATION_ENTRY structure
D3D12_SRV_DIMENSION enumeration
D3D12_STATE_OBJECT_CONFIG structure
D3D12_STATE_OBJECT_DESC structure
D3D12_STATE_OBJECT_FLAGS enumeration
D3D12_STATE_OBJECT_TYPE enumeration
D3D12_STATE_SUBOBJECT structure
D3D12_STATE_SUBOBJECT_TYPE enumeration
D3D12_STATIC_BORDER_COLOR enumeration
D3D12_STATIC_SAMPLER_DESC structure
D3D12_STENCIL_OP enumeration
D3D12_STREAM_OUTPUT_BUFFER_VIEW structure
D3D12_STREAM_OUTPUT_DESC structure
D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure
D3D12_SUBRESOURCE_DATA structure
D3D12_SUBRESOURCE_FOOTPRINT structure
D3D12_SUBRESOURCE_INFO structure
D3D12_SUBRESOURCE_RANGE_UINT64 structure
D3D12_SUBRESOURCE_TILING structure
D3D12_TEX1D_ARRAY_DSV structure
D3D12_TEX1D_ARRAY_RTV structure
D3D12_TEX1D_ARRAY_SRV structure
D3D12_TEX1D_ARRAY_UAV structure
D3D12_TEX1D_DSV structure
D3D12_TEX1D_RTV structure
D3D12_TEX1D_SRV structure
D3D12_TEX1D_UAV structure
D3D12_TEX2D_ARRAY_DSV structure

D3D12_TEX2D_ARRAY_RTV structure
D3D12_TEX2D_ARRAY_SRV structure
D3D12_TEX2D_ARRAY_UAV structure
D3D12_TEX2D_DSV structure
D3D12_TEX2D_RTV structure
D3D12_TEX2D_SRV structure
D3D12_TEX2D_UAV structure
D3D12_TEX2DMS_ARRAY_DSV structure
D3D12_TEX2DMS_ARRAY_RTV structure
D3D12_TEX2DMS_ARRAY_SRV structure
D3D12_TEX2DMS_DSV structure
D3D12_TEX2DMS_RTV structure
D3D12_TEX2DMS_SRV structure
D3D12_TEX3D_RTV structure
D3D12_TEX3D_SRV structure
D3D12_TEX3D_UAV structure
D3D12_TEXCUBE_ARRAY_SRV structure
D3D12_TEXCUBE_SRV structure
D3D12_TEXTURE_ADDRESS_MODE enumeration
D3D12_TEXTURE_COPY_LOCATION structure
D3D12_TEXTURE_COPY_TYPE enumeration
D3D12_TEXTURE_LAYOUT enumeration
D3D12_TILE_COPY_FLAGS enumeration
D3D12_TILE_MAPPING_FLAGS enumeration
D3D12_TILE_RANGE_FLAGS enumeration
D3D12_TILE_REGION_SIZE structure
D3D12_TILE_SHAPE structure
D3D12_TILED_RESOURCE_COORDINATE structure
D3D12_TILED_RESOURCES_TIER enumeration
D3D12_UAV_DIMENSION enumeration
D3D12_UNORDERED_ACCESS_VIEW_DESC structure
D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure
D3D12_VERSIONED_ROOT_SIGNATURE_DESC structure
D3D12_VERTEX_BUFFER_VIEW structure
D3D12_VIEW_INSTANCE_LOCATION structure
D3D12_VIEW_INSTANCING_DESC structure
D3D12_VIEW_INSTANCING_FLAGS enumeration
D3D12_VIEW_INSTANCING_TIER enumeration
D3D12_VIEWPORT structure

D3D12_WRITEBUFFERIMMEDIATE_MODE enumeration
D3D12_WRITEBUFFERIMMEDIATE_PARAMETER structure
D3D12CreateDevice function
D3D12CreateRootSignatureDeserializer function
D3D12CreateVersionedRootSignatureDeserializer function
D3D12EnableExperimentalFeatures function
D3D12GetDebugInterface function
D3D12SerializeRootSignature function
D3D12SerializeVersionedRootSignature function
ID3D12CommandAllocator interface
 Overview
 ID3D12CommandAllocator::Reset method
ID3D12CommandList interface
 Overview
 ID3D12CommandList::GetType method
ID3D12CommandQueue interface
 Overview
 ID3D12CommandQueue::BeginEvent method
 ID3D12CommandQueue::CopyTileMappings method
 ID3D12CommandQueue::EndEvent method
 ID3D12CommandQueue::ExecuteCommandLists method
 ID3D12CommandQueue::GetClockCalibration method
 ID3D12CommandQueue::GetDesc method
 ID3D12CommandQueue::GetTimestampFrequency method
 ID3D12CommandQueue::SetMarker method
 ID3D12CommandQueue::Signal method
 ID3D12CommandQueue::UpdateTileMappings method
 ID3D12CommandQueue::Wait method
ID3D12CommandSignature interface
ID3D12DescriptorHeap interface
 Overview
 ID3D12DescriptorHeap::GetCPUDescriptorHandleForHeapStart method
 ID3D12DescriptorHeap::GetDesc method
 ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart method
ID3D12Device interface
 Overview
 ID3D12Device::CheckFeatureSupport method
 ID3D12Device::CopyDescriptors method
 ID3D12Device::CopyDescriptorsSimple method

[ID3D12Device::CreateCommandAllocator](#) method
[ID3D12Device::CreateCommandList](#) method
[ID3D12Device::CreateCommandQueue](#) method
[ID3D12Device::CreateCommandSignature](#) method
[ID3D12Device::CreateCommittedResource](#) method
[ID3D12Device::CreateComputePipelineState](#) method
[ID3D12Device::CreateConstantBufferView](#) method
[ID3D12Device::CreateDepthStencilView](#) method
[ID3D12Device::CreateDescriptorHeap](#) method
[ID3D12Device::CreateFence](#) method
[ID3D12Device::CreateGraphicsPipelineState](#) method
[ID3D12Device::CreateHeap](#) method
[ID3D12Device::CreatePlacedResource](#) method
[ID3D12Device::CreateQueryHeap](#) method
[ID3D12Device::CreateRenderTargetView](#) method
[ID3D12Device::CreateReservedResource](#) method
[ID3D12Device::CreateRootSignature](#) method
[ID3D12Device::CreateSampler](#) method
[ID3D12Device::CreateShaderResourceView](#) method
[ID3D12Device::CreateSharedHandle](#) method
[ID3D12Device::CreateUnorderedAccessView](#) method
[ID3D12Device::Evict](#) method
[ID3D12Device::GetAdapterLuid](#) method
[ID3D12Device::GetCopyableFootprints](#) method
[ID3D12Device::GetCustomHeapProperties](#) method
[ID3D12Device::GetDescriptorHandleIncrementSize](#) method
[ID3D12Device::GetDeviceRemovedReason](#) method
[ID3D12Device::GetNodeCount](#) method
[ID3D12Device::GetResourceAllocationInfo](#) method
[ID3D12Device::GetResourceTiling](#) method
[ID3D12Device::MakeResident](#) method
[ID3D12Device::OpenSharedHandle](#) method
[ID3D12Device::OpenSharedHandleByName](#) method
[ID3D12Device::SetStablePowerState](#) method

ID3D12Device1 interface

[Overview](#)

[ID3D12Device1::CreatePipelineLibrary](#) method
[ID3D12Device1::SetEventOnMultipleFenceCompletion](#) method
[ID3D12Device1::SetResidencyPriority](#) method

ID3D12Device2 interface

[Overview](#)

[ID3D12Device2::CreatePipelineState method](#)

ID3D12Device3 interface

[Overview](#)

[ID3D12Device3::EnqueueMakeResident method](#)

[ID3D12Device3::OpenExistingHeapFromAddress method](#)

[ID3D12Device3::OpenExistingHeapFromFileMapping method](#)

ID3D12Device4 interface

[Overview](#)

[ID3D12Device4::CreateCommandList1 method](#)

[ID3D12Device4::CreateCommittedResource1 method](#)

[ID3D12Device4::CreateHeap1 method](#)

[ID3D12Device4::CreateProtectedResourceSession method](#)

[ID3D12Device4::CreateReservedResource1 method](#)

[ID3D12Device4::GetResourceAllocationInfo1 method](#)

ID3D12Device5 interface

[Overview](#)

[ID3D12Device5::CheckDriverMatchingIdentifier method](#)

[ID3D12Device5::CreateLifetimeTracker method](#)

[ID3D12Device5::CreateMetaCommand method](#)

[ID3D12Device5::CreateStateObject method](#)

[ID3D12Device5::EnumerateMetaCommandParameters method](#)

[ID3D12Device5::EnumerateMetaCommands method](#)

[ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo method](#)

[ID3D12Device5::RemoveDevice method](#)

ID3D12Device6 interface

[Overview](#)

[ID3D12Device6::SetBackgroundProcessingMode method](#)

ID3D12DeviceChild interface

[Overview](#)

[ID3D12DeviceChild::GetDevice method](#)

ID3D12DeviceRemovedExtendedData interface

[Overview](#)

[ID3D12DeviceRemovedExtendedData::GetAutoBreadcrumbsOutput method](#)

[ID3D12DeviceRemovedExtendedData::GetPageFaultAllocationOutput method](#)

ID3D12DeviceRemovedExtendedDataSettings interface

[Overview](#)

[ID3D12DeviceRemovedExtendedDataSettings::SetAutoBreadcrumbsEnablement method](#)

[ID3D12DeviceRemovedExtendedDataSettings::SetPageFaultEnablement](#) method

[ID3D12DeviceRemovedExtendedDataSettings::SetWatsonDumpEnablement](#) method

ID3D12Fence interface

[Overview](#)

[ID3D12Fence::GetCompletedValue](#) method

[ID3D12Fence::SetEventOnCompletion](#) method

[ID3D12Fence::Signal](#) method

ID3D12Fence1 interface

[Overview](#)

[ID3D12Fence1::GetCreationFlags](#) method

ID3D12GraphicsCommandList interface

[Overview](#)

[ID3D12GraphicsCommandList::BeginEvent](#) method

[ID3D12GraphicsCommandList::BeginQuery](#) method

[ID3D12GraphicsCommandList::ClearDepthStencilView](#) method

[ID3D12GraphicsCommandList::ClearRenderTargetView](#) method

[ID3D12GraphicsCommandList::ClearState](#) method

[ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat](#) method

[ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint](#) method

[ID3D12GraphicsCommandList::Close](#) method

[ID3D12GraphicsCommandList::CopyBufferRegion](#) method

[ID3D12GraphicsCommandList::CopyResource](#) method

[ID3D12GraphicsCommandList::CopyTextureRegion](#) method

[ID3D12GraphicsCommandList::CopyTiles](#) method

[ID3D12GraphicsCommandList::DiscardResource](#) method

[ID3D12GraphicsCommandList::Dispatch](#) method

[ID3D12GraphicsCommandList::DrawIndexedInstanced](#) method

[ID3D12GraphicsCommandList::DrawInstanced](#) method

[ID3D12GraphicsCommandList::EndEvent](#) method

[ID3D12GraphicsCommandList::EndQuery](#) method

[ID3D12GraphicsCommandList::ExecuteBundle](#) method

[ID3D12GraphicsCommandList::ExecuteIndirect](#) method

[ID3D12GraphicsCommandList::IASetIndexBuffer](#) method

[ID3D12GraphicsCommandList::IASetPrimitiveTopology](#) method

[ID3D12GraphicsCommandList::IASetVertexBuffers](#) method

[ID3D12GraphicsCommandList::OMSetBlendFactor](#) method

[ID3D12GraphicsCommandList::OMSetRenderTargets](#) method

[ID3D12GraphicsCommandList::OMSetStencilRef](#) method

[ID3D12GraphicsCommandList::Reset](#) method

[ID3D12GraphicsCommandList::ResolveQueryData](#) method
[ID3D12GraphicsCommandList::ResolveSubresource](#) method
[ID3D12GraphicsCommandList::ResourceBarrier](#) method
[ID3D12GraphicsCommandList::RSSetScissorRects](#) method
[ID3D12GraphicsCommandList::RSSetViewports](#) method
[ID3D12GraphicsCommandList::SetComputeRoot32BitConstant](#) method
[ID3D12GraphicsCommandList::SetComputeRoot32BitConstants](#) method
[ID3D12GraphicsCommandList::SetComputeRootConstantBufferView](#) method
[ID3D12GraphicsCommandList::SetComputeRootDescriptorTable](#) method
[ID3D12GraphicsCommandList::SetComputeRootShaderResourceView](#) method
[ID3D12GraphicsCommandList::SetComputeRootSignature](#) method
[ID3D12GraphicsCommandList::SetComputeRootUnorderedAccessView](#) method
[ID3D12GraphicsCommandList::SetDescriptorHeaps](#) method
[ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant](#) method
[ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants](#) method
[ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView](#) method
[ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable](#) method
[ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView](#) method
[ID3D12GraphicsCommandList::SetGraphicsRootSignature](#) method
[ID3D12GraphicsCommandList::SetGraphicsRootUnorderedAccessView](#) method
[ID3D12GraphicsCommandList::SetMarker](#) method
[ID3D12GraphicsCommandList::SetPipelineState](#) method
[ID3D12GraphicsCommandList::SetPredication](#) method
[ID3D12GraphicsCommandList::SOSetTargets](#) method

ID3D12GraphicsCommandList1 interface

Overview

[ID3D12GraphicsCommandList1::AtomicCopyBufferUINT](#) method
[ID3D12GraphicsCommandList1::AtomicCopyBufferUINT64](#) method
[ID3D12GraphicsCommandList1::OMSetDepthBounds](#) method
[ID3D12GraphicsCommandList1::ResolveSubresourceRegion](#) method
[ID3D12GraphicsCommandList1::SetSamplePositions](#) method
[ID3D12GraphicsCommandList1::SetViewInstanceMask](#) method

ID3D12GraphicsCommandList2 interface

Overview

[ID3D12GraphicsCommandList2::WriteBufferImmediate](#) method

ID3D12GraphicsCommandList3 interface

Overview

[ID3D12GraphicsCommandList3::SetProtectedResourceSession](#) method

ID3D12GraphicsCommandList4 interface

Overview

[ID3D12GraphicsCommandList4::BeginRenderPass method](#)
[ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure method](#)
[ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure method](#)
[ID3D12GraphicsCommandList4::DispatchRays method](#)
[ID3D12GraphicsCommandList4::EmitRaytracingAccelerationStructurePostbuildInfo method](#)
[ID3D12GraphicsCommandList4::EndRenderPass method](#)
[ID3D12GraphicsCommandList4::ExecuteMetaCommand method](#)
[ID3D12GraphicsCommandList4::InitializeMetaCommand method](#)
[ID3D12GraphicsCommandList4::SetPipelineState1 method](#)

ID3D12GraphicsCommandList5 interface

Overview

[ID3D12GraphicsCommandList5::RSSetShadingRate method](#)
[ID3D12GraphicsCommandList5::RSSetShadingRateImage method](#)

ID3D12Heap interface

Overview

[ID3D12Heap::GetDesc method](#)

ID3D12LifetimeOwner interface

Overview

[ID3D12LifetimeOwner::LifetimeStateChanged method](#)

ID3D12LifetimeTracker interface

Overview

[ID3D12LifetimeTracker::DestroyOwnedObject method](#)

ID3D12MetaCommand interface

Overview

[ID3D12MetaCommand::GetRequiredParameterResourceSize method](#)

ID3D12Object interface

Overview

[ID3D12Object::GetPrivateData method](#)

[ID3D12Object::SetName method](#)

[ID3D12Object::SetPrivateData method](#)

[ID3D12Object::SetPrivateDataInterface method](#)

ID3D12Pageable interface

ID3D12PipelineLibrary interface

Overview

[ID3D12PipelineLibrary::GetSerializedSize method](#)

[ID3D12PipelineLibrary::LoadComputePipeline method](#)

[ID3D12PipelineLibrary::LoadGraphicsPipeline method](#)

[ID3D12PipelineLibrary::Serialize method](#)

ID3D12PipelineLibrary::StorePipeline method

ID3D12PipelineLibrary1 interface

- Overview

ID3D12PipelineLibrary1::LoadPipeline method

ID3D12PipelineState interface

- Overview

ID3D12PipelineState::GetCachedBlob method

ID3D12ProtectedResourceSession interface

- Overview

ID3D12ProtectedResourceSession::GetDesc method

ID3D12ProtectedSession interface

- Overview

ID3D12ProtectedSession::GetSessionStatus method

ID3D12ProtectedSession::GetStatusFence method

ID3D12QueryHeap interface

ID3D12Resource interface

- Overview

ID3D12Resource::GetDesc method

ID3D12Resource::GetGPUVirtualAddress method

ID3D12Resource::GetHeapProperties method

ID3D12Resource::Map method

ID3D12Resource::ReadFromSubresource method

ID3D12Resource::Unmap method

ID3D12Resource::WriteToSubresource method

ID3D12RootSignature interface

ID3D12RootSignatureDeserializer interface

- Overview

ID3D12RootSignatureDeserializer::GetRootSignatureDesc method

ID3D12StateObject interface

ID3D12StateObjectProperties interface

- Overview

ID3D12StateObjectProperties::GetPipelineStackSize method

ID3D12StateObjectProperties::GetShaderIdentifier method

ID3D12StateObjectProperties::GetShaderStackSize method

ID3D12StateObjectProperties::SetPipelineStackSize method

ID3D12Tools interface

- Overview

ID3D12Tools::EnableShaderInstrumentation method

ID3D12Tools::ShaderInstrumentationEnabled method

ID3D12VersionedRootSignatureDeserializer interface

[Overview](#)

[ID3D12VersionedRootSignatureDeserializer::GetRootSignatureDescAtVersion method](#)

[ID3D12VersionedRootSignatureDeserializer::GetUnconvertedRootSignatureDesc method](#)

D3d12sdklayers.h

[Overview](#)

[D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS structure](#)

[D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE enumeration](#)

[D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS structure](#)

[D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR structure](#)

[D3D12_DEBUG_DEVICE_PARAMETER_TYPE enumeration](#)

[D3D12_DEBUG_FEATURE enumeration](#)

[D3D12_GPU_BASED_VALIDATION_FLAGS enumeration](#)

[D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS enumeration](#)

[D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE enumeration](#)

[D3D12_INFO_QUEUE_FILTER structure](#)

[D3D12_INFO_QUEUE_FILTER_DESC structure](#)

[D3D12_MESSAGE structure](#)

[D3D12_MESSAGE_CATEGORY enumeration](#)

[D3D12_MESSAGE_ID enumeration](#)

[D3D12_MESSAGE_SEVERITY enumeration](#)

[D3D12_RLDO_FLAGS enumeration](#)

ID3D12Debug interface

[Overview](#)

[ID3D12Debug::EnableDebugLayer method](#)

ID3D12Debug1 interface

[Overview](#)

[ID3D12Debug1::EnableDebugLayer method](#)

[ID3D12Debug1::SetEnableGPUBasedValidation method](#)

[ID3D12Debug1::SetEnableSynchronizedCommandQueueValidation method](#)

ID3D12Debug2 interface

[Overview](#)

[ID3D12Debug2::SetGPUBasedValidationFlags method](#)

ID3D12Debug3 interface

[Overview](#)

[ID3D12Debug3::SetEnableGPUBasedValidation method](#)

[ID3D12Debug3::SetEnableSynchronizedCommandQueueValidation method](#)

[ID3D12Debug3::SetGPUBasedValidationFlags method](#)

ID3D12DebugCommandList interface

[Overview](#)

[ID3D12DebugCommandList::AssertResourceState method](#)

[ID3D12DebugCommandList::GetFeatureMask method](#)

[ID3D12DebugCommandList::SetFeatureMask method](#)

[ID3D12DebugCommandList1 interface](#)

[Overview](#)

[ID3D12DebugCommandList1::AssertResourceState method](#)

[ID3D12DebugCommandList1::GetDebugParameter method](#)

[ID3D12DebugCommandList1::SetDebugParameter method](#)

[ID3D12DebugCommandQueue interface](#)

[Overview](#)

[ID3D12DebugCommandQueue::AssertResourceState method](#)

[ID3D12DebugDevice interface](#)

[Overview](#)

[ID3D12DebugDevice::GetFeatureMask method](#)

[ID3D12DebugDevice::ReportLiveDeviceObjects method](#)

[ID3D12DebugDevice::SetFeatureMask method](#)

[ID3D12DebugDevice1 interface](#)

[Overview](#)

[ID3D12DebugDevice1::GetDebugParameter method](#)

[ID3D12DebugDevice1::ReportLiveDeviceObjects method](#)

[ID3D12DebugDevice1::SetDebugParameter method](#)

[ID3D12InfoQueue interface](#)

[Overview](#)

[ID3D12InfoQueue::AddApplicationMessage method](#)

[ID3D12InfoQueue::AddMessage method](#)

[ID3D12InfoQueue::AddRetrievalFilterEntries method](#)

[ID3D12InfoQueue::AddStorageFilterEntries method](#)

[ID3D12InfoQueue::ClearRetrievalFilter method](#)

[ID3D12InfoQueue::ClearStorageFilter method](#)

[ID3D12InfoQueue::ClearStoredMessages method](#)

[ID3D12InfoQueue::GetBreakOnCategory method](#)

[ID3D12InfoQueue::GetBreakOnID method](#)

[ID3D12InfoQueue::GetBreakOnSeverity method](#)

[ID3D12InfoQueue::GetMessage method](#)

[ID3D12InfoQueue::GetMessageCountLimit method](#)

[ID3D12InfoQueue::GetMuteDebugOutput method](#)

[ID3D12InfoQueue::GetNumMessagesAllowedByStorageFilter method](#)

[ID3D12InfoQueue::GetNumMessagesDeniedByStorageFilter method](#)

[ID3D12InfoQueue::GetNumMessagesDiscardedByMessageCountLimit](#) method
[ID3D12InfoQueue::GetNumStoredMessages](#) method
[ID3D12InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter](#) method
[ID3D12InfoQueue::GetRetrievalFilter](#) method
[ID3D12InfoQueue::GetRetrievalFilterStackSize](#) method
[ID3D12InfoQueue::GetStorageFilter](#) method
[ID3D12InfoQueue::GetStorageFilterStackSize](#) method
[ID3D12InfoQueue::PopRetrievalFilter](#) method
[ID3D12InfoQueue::PopStorageFilter](#) method
[ID3D12InfoQueue::PushCopyOfRetrievalFilter](#) method
[ID3D12InfoQueue::PushCopyOfStorageFilter](#) method
[ID3D12InfoQueue::PushEmptyRetrievalFilter](#) method
[ID3D12InfoQueue::PushEmptyStorageFilter](#) method
[ID3D12InfoQueue::PushRetrievalFilter](#) method
[ID3D12InfoQueue::PushStorageFilter](#) method
[ID3D12InfoQueue::SetBreakOnCategory](#) method
[ID3D12InfoQueue::SetBreakOnID](#) method
[ID3D12InfoQueue::SetBreakOnSeverity](#) method
[ID3D12InfoQueue::SetMessageCountLimit](#) method
[ID3D12InfoQueue::SetMuteDebugOutput](#) method

ID3D12SharingContract interface

[Overview](#)
[ID3D12SharingContract::Present](#) method
[ID3D12SharingContract::SharedFenceSignal](#) method

D3d12shader.h

[Overview](#)
[D3D12_FUNCTION_DESC](#) structure
[D3D12_LIBRARY_DESC](#) structure
[D3D12_PARAMETER_DESC](#) structure
[D3D12_SHADER_BUFFER_DESC](#) structure
[D3D12_SHADER_DESC](#) structure
[D3D12_SHADER_INPUT_BIND_DESC](#) structure
[D3D12_SHADER_TYPE_DESC](#) structure
[D3D12_SHADER_VARIABLE_DESC](#) structure
[D3D12_SHADER_VERSION_TYPE](#) enumeration
[D3D12_SIGNATURE_PARAMETER_DESC](#) structure

ID3D12FunctionParameterReflection interface

[Overview](#)
[ID3D12FunctionParameterReflection::GetDesc](#) method

ID3D12FunctionReflection interface

Overview

[ID3D12FunctionReflection::GetConstantBufferByIndex method](#)

[ID3D12FunctionReflection::GetConstantBufferByName method](#)

[ID3D12FunctionReflection::GetDesc method](#)

[ID3D12FunctionReflection::GetFunctionParameter method](#)

[ID3D12FunctionReflection::GetResourceBindingDesc method](#)

[ID3D12FunctionReflection::GetResourceBindingDescByName method](#)

[ID3D12FunctionReflection::GetVariableByName method](#)

ID3D12LibraryReflection interface

Overview

[ID3D12LibraryReflection::GetDesc method](#)

[ID3D12LibraryReflection::GetFunctionByIndex method](#)

ID3D12ShaderReflection interface

Overview

[ID3D12ShaderReflection::GetBitwiseInstructionCount method](#)

[ID3D12ShaderReflection::GetConstantBufferByIndex method](#)

[ID3D12ShaderReflection::GetConstantBufferByName method](#)

[ID3D12ShaderReflection::GetConversionInstructionCount method](#)

[ID3D12ShaderReflection::GetDesc method](#)

[ID3D12ShaderReflection::GetGSInputPrimitive method](#)

[ID3D12ShaderReflection::GetInputParameterDesc method](#)

[ID3D12ShaderReflection::GetMinFeatureLevel method](#)

[ID3D12ShaderReflection::GetMovInstructionCount method](#)

[ID3D12ShaderReflection::GetMovInstructionCount method](#)

[ID3D12ShaderReflection::GetNumInterfaceSlots method](#)

[ID3D12ShaderReflection::GetOutputParameterDesc method](#)

[ID3D12ShaderReflection::GetPatchConstantParameterDesc method](#)

[ID3D12ShaderReflection::GetRequiresFlags method](#)

[ID3D12ShaderReflection::GetResourceBindingDesc method](#)

[ID3D12ShaderReflection::GetResourceBindingDescByName method](#)

[ID3D12ShaderReflection::GetThreadGroupSize method](#)

[ID3D12ShaderReflection::GetVariableByName method](#)

[ID3D12ShaderReflection::IsSampleFrequencyShader method](#)

ID3D12ShaderReflectionConstantBuffer interface

Overview

[ID3D12ShaderReflectionConstantBuffer::GetDesc method](#)

[ID3D12ShaderReflectionConstantBuffer::GetVariableByIndex method](#)

[ID3D12ShaderReflectionConstantBuffer::GetVariableByName method](#)

ID3D12ShaderReflectionType interface

Overview

[ID3D12ShaderReflectionType::GetBaseClass](#) method

[ID3D12ShaderReflectionType::GetDesc](#) method

[ID3D12ShaderReflectionType::GetInterfaceByIndex](#) method

[ID3D12ShaderReflectionType::GetMemberTypeByIndex](#) method

[ID3D12ShaderReflectionType::GetMemberTypeByName](#) method

[ID3D12ShaderReflectionType::GetMemberTypeName](#) method

[ID3D12ShaderReflectionType::GetNumInterfaces](#) method

[ID3D12ShaderReflectionType::GetSubType](#) method

[ID3D12ShaderReflectionType::ImplementsInterface](#) method

[ID3D12ShaderReflectionType::IsEqual](#) method

[ID3D12ShaderReflectionType::IsOfType](#) method

ID3D12ShaderReflectionVariable interface

Overview

[ID3D12ShaderReflectionVariable::GetBuffer](#) method

[ID3D12ShaderReflectionVariable::GetDesc](#) method

[ID3D12ShaderReflectionVariable::GetInterfaceSlot](#) method

[ID3D12ShaderReflectionVariable::GetType](#) method

Directml.h

Overview

[DML_ACTIVATION_ELU_OPERATOR_DESC](#) structure

[DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC](#) structure

[DML_ACTIVATION_HARDMAX_OPERATOR_DESC](#) structure

[DML_ACTIVATION_IDENTITY_OPERATOR_DESC](#) structure

[DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC](#) structure

[DML_ACTIVATION_LINEAR_OPERATOR_DESC](#) structure

[DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC](#) structure

[DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC](#) structure

[DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC](#) structure

[DML_ACTIVATION_RELU_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SHRINK_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SIGMOID_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SOFTMAX_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC](#) structure

[DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC](#) structure

[DML_ACTIVATION_TANH_OPERATOR_DESC](#) structure

DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC structure
DML_AVERAGE_POOLING_OPERATOR_DESC structure
DML_BATCH_NORMALIZATION_OPERATOR_DESC structure
DML_BINDING_DESC structure
DML_BINDING_PROPERTIES structure
DML_BINDING_TABLE_DESC structure
DML_BINDING_TYPE enumeration
DML_BUFFER_ARRAY_BINDING structure
DML_BUFFER_BINDING structure
DML_BUFFER_TENSOR_DESC structure
DML_CAST_OPERATOR_DESC structure
DML_CONVOLUTION_DIRECTION enumeration
DML_CONVOLUTION_MODE enumeration
DML_CONVOLUTION_OPERATOR_DESC structure
DML_CREATE_DEVICE_FLAGS enumeration
DML_DEPTH_TO_SPACE_OPERATOR_DESC structure
DML_DIAGONAL_MATRIX_OPERATOR_DESC structure
DML_ELEMENT_WISE_ABS_OPERATOR_DESC structure
DML_ELEMENT_WISE_ACOS_OPERATOR_DESC structure
DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC structure
DML_ELEMENT_WISE_ADD_OPERATOR_DESC structure
DML_ELEMENT_WISE_ADD1_OPERATOR_DESC structure
DML_ELEMENT_WISE_ASIN_OPERATOR_DESC structure
DML_ELEMENT_WISE_ASINH_OPERATOR_DESC structure
DML_ELEMENT_WISE_ATAN_OPERATOR_DESC structure
DML_ELEMENT_WISE_ATANH_OPERATOR_DESC structure
DML_ELEMENT_WISE_CEIL_OPERATOR_DESC structure
DML_ELEMENT_WISE_CLIP_OPERATOR_DESC structure
DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC structure
DML_ELEMENT_WISE_COS_OPERATOR_DESC structure
DML_ELEMENT_WISE_COSH_OPERATOR_DESC structure
DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC structure
DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC structure
DML_ELEMENT_WISE_ERF_OPERATOR_DESC structure
DML_ELEMENT_WISE_EXP_OPERATOR_DESC structure
DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC structure
DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC structure
DML_ELEMENT_WISE_IF_OPERATOR_DESC structure
DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC structure

DML_ELEMENT_WISE_LOG_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC structure
DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC structure
DML_ELEMENT_WISE_MAX_OPERATOR_DESC structure
DML_ELEMENT_WISE_MEAN_OPERATOR_DESC structure
DML_ELEMENT_WISE_MIN_OPERATOR_DESC structure
DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC structure
DML_ELEMENT_WISE_POW_OPERATOR_DESC structure
DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC structure
DML_ELEMENT_WISE_RECIP_OPERATOR_DESC structure
DML_ELEMENT_WISE_SIGN_OPERATOR_DESC structure
DML_ELEMENT_WISE_SIN_OPERATOR_DESC structure
DML_ELEMENT_WISE_SINH_OPERATOR_DESC structure
DML_ELEMENT_WISE_SQRT_OPERATOR_DESC structure
DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC structure
DML_ELEMENT_WISE_TAN_OPERATOR_DESC structure
DML_ELEMENT_WISE_TANH_OPERATOR_DESC structure
DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC structure
DML_EXECUTION_FLAGS enumeration
DML_FEATURE enumeration
DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT structure
DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT structure
DML_GATHER_OPERATOR_DESC structure
DML_GEMM_OPERATOR_DESC structure
DML_GRU_OPERATOR_DESC structure
DML_INTERPOLATION_MODE enumeration
DML_JOIN_OPERATOR_DESC structure
DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC structure
DML_LP_NORMALIZATION_OPERATOR_DESC structure
DML_LP_POOLING_OPERATOR_DESC structure
DML_LSTM_OPERATOR_DESC structure
DML_MATRIX_TRANSFORM enumeration
DML_MAX_POOLING_OPERATOR_DESC structure
DML_MAX_POOLING1_OPERATOR_DESC structure

DML_MAX_UNPOOLING_OPERATOR_DESC structure
DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC structure
DML_ONE_HOT_OPERATOR_DESC structure
DML_OPERATOR_DESC structure
DML_OPERATOR_TYPE enumeration
DML_PADDING_MODE enumeration
DML_PADDING_OPERATOR_DESC structure
DML_RECURRENT_NETWORK_DIRECTION enumeration
DML_REDUCE_FUNCTION enumeration
DML_REDUCE_OPERATOR_DESC structure
DML_RESAMPLE_OPERATOR_DESC structure
DML_RNN_OPERATOR_DESC structure
DML_ROI_POOLING_OPERATOR_DESC structure
DML_SCALE_BIAS structure
DML_SCATTER_OPERATOR_DESC structure
DML_SIZE_2D structure
DML_SLICE_OPERATOR_DESC structure
DML_SPACE_TO_DEPTH_OPERATOR_DESC structure
DML_SPLIT_OPERATOR_DESC structure
DML_TENSOR_DATA_TYPE enumeration
DML_TENSOR_DESC structure
DML_TENSOR_FLAGS enumeration
DML_TENSOR_TYPE enumeration
DML_TILE_OPERATOR_DESC structure
DML_TOP_K_OPERATOR_DESC structure
DML_UPSAMPLE_2D_OPERATOR_DESC structure
DML_VALUE_SCALE_2D_OPERATOR_DESC structure
DMLCreateDevice function
IDMLBindingTable interface

 Overview

 IDMLBindingTable::BindInputs method
 IDMLBindingTable::BindOutputs method
 IDMLBindingTable::BindPersistentResource method
 IDMLBindingTable::BindTemporaryResource method
 IDMLBindingTable::Reset method

IDMLCommandRecorder interface

 Overview

 IDMLCommandRecorder::RecordDispatch method

IDMLCompiledOperator interface

[IDMLDebugDevice interface](#)

[Overview](#)

[IDMLDebugDevice::SetMuteDebugOutput method](#)

[IDMLDevice interface](#)

[Overview](#)

[IDMLDevice::CheckFeatureSupport method](#)

[IDMLDevice::CompileOperator method](#)

[IDMLDevice::CreateBindingTable method](#)

[IDMLDevice::CreateCommandRecorder method](#)

[IDMLDevice::CreateOperator method](#)

[IDMLDevice::CreateOperatorInitializer method](#)

[IDMLDevice::Evict method](#)

[IDMLDevice::GetDeviceRemovedReason method](#)

[IDMLDevice::GetParentDevice method](#)

[IDMLDevice::MakeResident method](#)

[IDMLDeviceChild interface](#)

[Overview](#)

[IDMLDeviceChild::GetDevice method](#)

[IDMLDispatchable interface](#)

[Overview](#)

[IDMLDispatchable::GetBindingProperties method](#)

[IDMLObject interface](#)

[Overview](#)

[IDMLObject::GetPrivateData method](#)

[IDMLObject::SetName method](#)

[IDMLObject::SetPrivateData method](#)

[IDMLObject::SetPrivateDataInterface method](#)

[IDMLOperator interface](#)

[IDMLOperatorInitializer interface](#)

[Overview](#)

[IDMLOperatorInitializer::Reset method](#)

[IDMLPageable interface](#)

[Windows.graphics.holographic.interop.h](#)

[Overview](#)

[IHolographicCameraInterop interface](#)

[Overview](#)

[IHolographicCameraInterop::AcquireDirect3D12BufferResource method](#)

[IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout method](#)

[IHolographicCameraInterop::CreateDirect3D12BackBufferResource method](#)

[IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource](#) method

[IHolographicCameraInterop::UnacquireDirect3D12BufferResource](#) method

[IHolographicCameraRenderingParametersInterop](#) interface

[Overview](#)

[IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource](#) method

[IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData](#) method

[IHolographicQuadLayerInterop](#) interface

[Overview](#)

[IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource](#) method

[IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout](#) method

[IHolographicQuadLayerInterop::CreateDirect3D12ContentBufferResource](#) method

[IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource](#) method

[IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource](#) method

[IHolographicQuadLayerUpdateParametersInterop](#) interface

[Overview](#)

[IHolographicQuadLayerUpdateParametersInterop::CommitDirect3D12Resource](#) method

Direct3D 12 Graphics

6/30/2020 • 64 minutes to read • [Edit Online](#)

Overview of the Direct3D 12 Graphics technology.

To develop Direct3D 12 Graphics, you need these headers:

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

For programming guidance for this technology, see:

- [Direct3D 12 Graphics](#)

Enumerations

TITLE	DESCRIPTION
D3D_ROOT_SIGNATURE_VERSION	Specifies the version of root signature layout.
D3D_SHADER_MODEL	Specifies a shader model.
D3D12_BACKGROUND_PROCESSING_MODE	Defines constants that specify a level of dynamic optimization to apply to GPU work that's subsequently submitted.
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_UAV_FLAGS	Identifies unordered-access view options for a buffer resource.
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_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_QUEUE_FLAGS	Specifies flags to be used when creating a command queue.

TITLE	DESCRIPTION
D3D12_COMMAND_QUEUE_PRIORITY	Defines priority levels for a command queue.
D3D12_COMPARISON_FUNC	Specifies comparison options.
D3D12_CONSERVATIVE_RASTERIZATION_MODE	Identifies whether conservative rasterization is on or off.
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_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_FLAGS	Specifies depth-stencil view options.
D3D12_ELEMENTS_LAYOUT	Describes how the locations of elements are identified.

TITLE	DESCRIPTION
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_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_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_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.

TITLE	DESCRIPTION
D3D12_LOGIC_OP	Specifies logical operations to configure for a render target.
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_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_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_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.

TITLE	DESCRIPTION
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_IN_PUTS 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_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.

TITLE	DESCRIPTION
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_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_SERIALIZED_DATA_TYPE	Specifies the type of serialized data. Use a value from this enumeration when calling ID3D12Device5::CheckDriverMatchingIdentifier.
D3D12_SHADER_CACHE_SUPPORT_FLAGS	Describes the level of support for shader caching in the current graphics driver.
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_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.

TITLE	DESCRIPTION
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_COPY_TYPE	Specifies what type of texture copy is to take place.
D3D12_TEXTURE_LAYOUT	Specifies texture layout options.
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_UAV_DIMENSION	Identifies unordered-access view options.
D3D12_VIEW_INSTANCING_FLAGS	Specifies options for view instancing.
D3D12_VIEW_INSTANCING_TIER	Indicates the tier level at which view instancing is supported.
D3D12_WRITEBUFFERIMMEDIATE_MODE	Specifies the mode used by a WriteBufferImmediate operation.
DML_BINDING_TYPE	Defines constants that specify the nature of the resource(s) referred to by a binding description (a DML_BINDING_DESC structure).
DML_CONVOLUTION_DIRECTION	Defines constants that specify a direction for the DirectML convolution operator (as described by the DML_CONVOLUTION_OPERATOR_DESC structure).
DML_CONVOLUTION_MODE	Defines constants that specify a mode for the DirectML convolution operator (as described by the DML_CONVOLUTION_OPERATOR_DESC structure).
DML_CREATE_DEVICE_FLAGS	Supplies additional device creation options to DMLCreateDevice. Values can be bitwise OR'd together.
DML_EXECUTION_FLAGS	Supplies options to DirectML to control execution of operators. These flags can be bitwise OR'd together to specify multiple flags at once.

TITLE	DESCRIPTION
DML_FEATURE	Defines a set of optional features and capabilities that can be queried from the DirectML device.
DML_INTERPOLATION_MODE	Defines constants that specify a mode for the DirectML upsample 2-D operator (as described by the DML_UPSAMPLE_2D_OPERATOR_DESC structure).
DML_MATRIX_TRANSFORM	Defines constants that specify a matrix transform to be applied to a DirectML tensor.
DML_OPERATOR_TYPE	Defines the type of an operator description.
DML_PADDING_MODE	Defines constants that specify a mode for the DirectML pad operator (as described by the DML_PADDING_OPERATOR_DESC structure).
DML_RECURRENT_NETWORK_DIRECTION	Defines constants that specify a direction for a recurrent DirectML operator.
DML_REDUCE_FUNCTION	Defines constants that specify the specific reduction algorithm to use for the DirectML reduce operator (as described by the DML_REDUCE_OPERATOR_DESC structure).
DML_TENSOR_DATA_TYPE	Specifies the data type of the values in a tensor. DirectML operators may not support all data types; see the documentation for each specific operator to find which data types it supports.
DML_TENSOR_FLAGS	Specifies additional options in a tensor description. Values can be bitwise OR'd together.
DML_TENSOR_TYPE	Identifies a type of tensor description.

Functions

TITLE	DESCRIPTION
AcquireDirect3D12BufferResource	Acquires a Direct3D 12 buffer resource.
AcquireDirect3D12BufferResource	Acquires a Direct3D 12 buffer resource.
AcquireDirect3D12BufferResourceWithTimeout	Acquires a Direct3D 12 buffer resource, with an optional timeout.
AcquireDirect3D12BufferResourceWithTimeout	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.

TITLE	DESCRIPTION
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.
AddStorageFilterEntries	Add storage filters to the top of the storage-filter stack.
AssertResourceState	Checks whether a resource, or subresource, is in a specified state, or not.
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.
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.
BeginEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command queue.
BeginEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command list.
BeginQuery	Starts a query running.
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).
BindInputs	Binds a set of resources as input tensors.
BindOutputs	Binds a set of resources as output tensors.
BindPersistentResource	Binds a buffer as a persistent resource. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties.
BindTemporaryResource	Binds a buffer to use as temporary scratch memory. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties.
BuildRaytracingAccelerationStructure	Performs a raytracing acceleration structure build on the GPU and optionally outputs post-build information immediately after the build.

TITLE	DESCRIPTION
CheckDriverMatchingIdentifier	Reports the compatibility of serialized data, such as a serialized raytracing acceleration structure resulting from a call to <code>CopyRaytracingAccelerationStructure</code> with mode <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE</code> , with the current device/driver.
CheckFeatureSupport	Gets information about the features that are supported by the current graphics driver.
CheckFeatureSupport	Gets information about the optional features and capabilities that are supported by the DirectML device.
ClearDepthStencilView	Clears the depth-stencil resource.
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.
ClearState	Resets the state of a direct command list back to the state it was in when the command list was created.
ClearStorageFilter	Remove a storage filter from the top of the storage-filter stack.
ClearStoredMessages	Clear all messages from the message queue.
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.
CommitDirect3D12Resource	Commits a Direct3D 12 buffer for presentation on outputs associated with the HolographicCamera .
CommitDirect3D12Resource	Commits a Direct3D 12 buffer for presentation on outputs associated with any HolographicCamera to which the quad layer is attached.
CommitDirect3D12ResourceWithDepthData	Commits a Direct3D 12 buffer for presentation on outputs associated with the HolographicCamera .
CompileOperator	Compiles an operator into an object that can be dispatched to the GPU.
CopyBufferRegion	Copies a region of a buffer from one resource to another.
CopyDescriptors	Copies descriptors from a source to a destination.
CopyDescriptorsSimple	Copies descriptors from a source to a destination.

TITLE	DESCRIPTION
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.
CreateBindingTable	Creates a binding table, which is an object that can be used to bind resources (such as tensors) to the pipeline.
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.
CreateCommandRecorder	Creates a DirectML command recorder.
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.
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.

TITLE	DESCRIPTION
CreateDirect3D12HardwareProtectedBackBufferResource	Creates a Direct3D 12 resource for use as a content buffer for the camera, with optional hardware protection.
CreateDirect3D12HardwareProtectedContentBufferResource	Creates a Direct3D 12 resource for use as a content buffer for the camera, with optional hardware protection.
CreateFence	Creates a fence object.
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.
CreateOperator	Creates a DirectML operator.
CreateOperatorInitializer	Creates an object that can be used to initialize compiled operators.
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.
CreateProtectedResourceSession	Creates an object that represents a session for content protection.
CreateQueryHeap	Creates a query heap. A query heap contains an array of queries.
CreateRenderTargetView	Creates a render-target view for accessing resource data.
CreateReservedResource	Creates a resource that is reserved, and not yet mapped to any pages in a heap.
CreateReservedResource1	Creates a resource (optionally for a protected session) that is reserved, and not yet mapped to any pages in a heap.
CreateRootSignature	Creates a root signature layout.

TITLE	DESCRIPTION
CreateSampler	Create a sampler object that encapsulates sampling information for a texture.
CreateShaderResourceView	Creates a shader-resource view for accessing data in a resource.
CreateSharedHandle	Creates a shared handle to an heap, resource, or fence object.
CreateStateObject	Creates an ID3D12StateObject.
CreateUnorderedAccessView	Creates a view for unordered accessing.
CreateWrappedResource	This method creates D3D11 resources for use with D3D 11on12.
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.
D3D12CreateDevice	Creates a device that represents the display adapter.
D3D12CreateRootSignatureDeserializer	Deserializes a root signature so you can determine the layout definition (D3D12_ROOT_SIGNATURE_DESC).
D3D12CreateVersionedRootSignatureDeserializer	Generates an interface that can return the serialized data structure, via GetUnconvertedRootSignatureDesc.
D3D12EnableExperimentalFeatures	Enables a list of experimental features.
D3D12GetDebugInterface	Gets a debug interface.
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.
DiscardResource	Discards a resource.
Dispatch	Executes a command list from a thread group.
DispatchRays	Launch the threads of a ray generation shader.
DMLCreateDevice	Creates a DirectML device for a given Direct3D 12 device.
DrawIndexedInstanced	Draws indexed, instanced primitives.
DrawInstanced	Draws non-indexed, instanced primitives.

TITLE	DESCRIPTION
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.
EnableDebugLayer	Enables the debug layer.
EnableDebugLayer	Enables the debug layer.
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.
EndEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command list.
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.
Evict	Evicts one or more pageable objects from GPU memory. Also see IDMLDevice::MakeResident.
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.
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.
GetBaseClass	Gets an ID3D12ShaderReflectionType Interface interface containing the variable base class type.

TITLE	DESCRIPTION
GetBindingProperties	Retrieves the binding properties for a dispatchable object (an operator initializer, or a compiled operator).
GetBitwiseInstructionCount	Gets the number of bitwise instructions.
GetBreakOnCategory	Get a message category to break on when a message with that category passes through the storage filter.
GetBreakOnID	Get a message identifier to break on when a message with that identifier passes through the storage filter.
GetBreakOnSeverity	Get a message severity level to break on when a message with that severity level passes through the storage filter.
GetBuffer	Returns the ID3D12ShaderReflectionConstantBuffer of the present ID3D12ShaderReflectionVariable.
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.
GetConstantBufferByIndex	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.
GetConstantBufferByName	Gets a constant buffer by name.
GetConversionInstructionCount	Gets the number of conversion instructions.
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.
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.
GetCustomHeapProperties	Divulges the equivalent custom heap properties that are used for non-custom heap types, based on the adapter's architectural properties.
GetDebugParameter	Gets optional Command List Debug Layer settings.
GetDebugParameter	Gets optional device-wide Debug Layer settings.

TITLE	DESCRIPTION
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.
GetDesc	Gets the resource description.
GetDesc	Fills the parameter descriptor structure for the function's parameter.
GetDesc	Fills the function descriptor structure for the function.
GetDesc	Fills the library descriptor structure for the library reflection.
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.
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.
GetDevice	Retrieves the DirectML device that was used to create this object.
GetDeviceRemovedReason	Gets the reason that the device was removed.
GetDeviceRemovedReason	Retrieves the reason that the DirectML device was removed.
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	Gets the function reflector.
GetFunctionParameter	Gets the function parameter reflector.
GetGPUDescriptorHandleForHeapStart	Gets the GPU descriptor handle that represents the start of the heap.

TITLE	DESCRIPTION
GetGPUVirtualAddress	This method returns the GPU virtual address of a buffer resource.
GetGSInputPrimitive	Gets the geometry-shader input-primitive description.
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.
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.
GetMessageCountLimit	Get the maximum number of messages that can be added to the message queue.
GetMinFeatureLevel	Gets the minimum feature level.
GetMovcInstructionCount	Gets the number of Movc instructions.
GetMovlInstructionCount	Gets the number of Mov instructions.
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.
GetNumInterfaceSlots	Gets the number of interface slots in a shader.
GetNumMessagesAllowedByStorageFilter	Get the number of messages that were allowed to pass through a storage filter.
GetNumMessagesDeniedByStorageFilter	Get the number of messages that were denied passage through a storage filter.
GetNumMessagesDiscardedByMessageCountLimit	Get the number of messages that were discarded due to the message count limit.
GetNumStoredMessages	Get the number of messages currently stored in the message queue.

TITLE	DESCRIPTION
GetNumStoredMessagesAllowedByRetrievalFilter	Get the number of messages that are able to pass through a retrieval filter.
GetOutputParameterDesc	Gets an output-parameter description for a shader.
GetPageFaultAllocationOutput	Retrieves the Device Removed Extended Data (DRED) page fault data.
GetParentDevice	Retrieves the Direct3D 12 device that was used to create this DirectML device.
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.
GetPrivateData	Gets application-defined data from a DirectML device object.
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.
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.
GetResourceBindingDesc	Gets a description of how a resource is bound to a function.
GetResourceBindingDesc	Gets a description of how a resource is bound to a shader.
GetResourceBindingDescByName	Gets a description of how a resource is bound to a function.
GetResourceBindingDescByName	Gets a description of how a resource is bound to a shader.
GetResourceTiling	Gets info about how a tiled resource is broken into tiles.
GetRetrievalFilter	Get the retrieval filter at the top of the retrieval-filter stack.
GetRetrievalFilterStackSize	Get the size of the retrieval-filter stack in bytes.
GetRootSignatureDesc	Gets the layout of the root signature.

TITLE	DESCRIPTION
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 ID3D12Fence::GetCompletedValue), 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.
GetStorageFilterStackSize	Get the size of the storage-filter stack in bytes.
GetSubType	Gets the base class of a class.
GetThreadGroupSize	Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid.
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.
GetVariableByName	Gets a variable by name.
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.

TITLE	DESCRIPTION
IASetVertexBuffers	Sets a CPU descriptor handle for the vertex buffers.
ImplementsInterface	Indicates whether a class type implements an interface.
InitializeMetaCommand	Initializes the specified meta command.
IsEqual	Indicates whether two ID3D12ShaderReflectionType Interface pointers have the same underlying type.
IsOfType	Indicates whether a variable is of the specified type.
IsSampleFrequencyShader	Indicates whether a shader is a sample frequency shader.
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.
MakeResident	Causes one or more pageable objects to become resident in GPU memory. Also see IDMLDevice::Evict .
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	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.

TITLE	DESCRIPTION
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.
PopRetrievalFilter	Pop a retrieval filter from the top of the retrieval-filter stack.
PopStorageFilter	Pop a storage filter from the top of the storage-filter stack.
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.
PushCopyOfStorageFilter	Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack.
PushEmptyRetrievalFilter	Push an empty retrieval filter onto the retrieval-filter stack.
PushEmptyStorageFilter	Push an empty storage filter onto the storage-filter stack.
PushRetrievalFilter	Push a retrieval filter onto the retrieval-filter stack.
PushStorageFilter	Push a storage filter onto the storage-filter stack.
ReadFromSubresource	Uses the CPU to copy data from a subresource, enabling the CPU to read the contents of most textures with undefined layouts.
RecordDispatch	Records execution of a dispatchable object (an operator initializer, or a compiled operator) onto a command list.
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.

TITLE	DESCRIPTION
Reset	Resets a command list back to its initial state as if a new command list was just created.
Reset	Resets the binding table to wrap a new range of descriptors, potentially for a different operator or initializer. This allows dynamic reuse of the binding table.
Reset	Resets the initializer to handle initialization of a new set of operators.
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.
RSSetScissorRects	Binds an array of scissor rectangles to the rasterizer stage.
RSSetShadingRate	
RSSetShadingRateImage	
RSSetViewports	Bind an array of viewports to the rasterizer stage of the pipeline.
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.
SetBreakOnCategory	Set a message category to break on when a message with that category passes through the storage filter.
SetBreakOnID	Set a message identifier to break on when a message with that identifier passes through the storage filter.
SetBreakOnSeverity	Set a message severity level to break on when a message with that severity level passes through the storage filter.
SetComputeRoot32BitConstant	Sets a constant in the compute root signature.
SetComputeRoot32BitConstants	Sets a group of constants in the compute root signature.

TITLE	DESCRIPTION
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.
SetDescriptorHeaps	Changes the currently bound descriptor heaps that are associated with a command list.
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.
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.
SetGPUBasedValidationFlags	This method configures the level of GPU-based validation that the debug device is to perform at runtime.
SetGPUBasedValidationFlags	This method configures the level of GPU-based validation that the debug device is to perform at runtime.
SetGraphicsRoot32BitConstant	Sets a constant in the graphics root signature.

TITLE	DESCRIPTION
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.
SetMarker	Not intended to be called directly. Use the PIX event runtime to insert events into a command list.
SetMessageCountLimit	Set the maximum number of messages that can be added to the message queue.
SetMuteDebugOutput	Set a boolean that turns the debug output on or off.
SetMuteDebugOutput	Determine whether to mute DirectML from sending messages to the ID3D12InfoQueue.
SetName	Associates a name with the device object. This name is for use in debug diagnostics and tools.
SetName	Associates a name with the DirectML 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.
SetPrivateData	Sets application-defined data to a DirectML device object, and associates that data with an application-defined GUID.

TITLE	DESCRIPTION
SetPrivateDataInterface	Associates an IUnknown-derived interface with the device object and associates that interface with an application-defined GUID.
SetPrivateDataInterface	Associates an IUnknown-derived interface with the DirectML 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.
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.
ShaderInstrumentationEnabled	Determines whether shader instrumentation is enabled.
SharedFenceSignal	Signals a shared fence between the D3D layers and diagnostics tools.
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.
UnacquireDirect3D12BufferResource	Un-acquires a Direct3D 12 buffer resource.
UnacquireDirect3D12BufferResource	Un-acquires a Direct3D 12 buffer resource.
Unmap	Invalidates the CPU pointer to the specified subresource in the resource. Unmap also flushes the CPU cache, when necessary, so that GPU reads to this address reflect any modifications made by the CPU.
UpdateTileMappings	Updates mappings of tile locations in reserved resources to memory locations in a resource heap.

TITLE	DESCRIPTION
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.
WriteToSubresource	Uses the CPU to copy data into a subresource, enabling the CPU to modify the contents of most textures with undefined layouts.

Interfaces

TITLE	DESCRIPTION
ID3D11On12Device	Handles the creation, wrapping, and releasing of D3D11 resources for Direct3D11on12.
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.
ID3D12Debug3	Adds configurable levels of GPU-based validation to the debug layer.
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.

TITLE	DESCRIPTION
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.
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.
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.
ID3D12DeviceRemovedExtendedDataSettings	This interface controls Device Removed Extended Data (DRED) settings.
ID3D12Fence	Represents a fence, an object used for synchronization of the CPU and one or more GPUs.
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.

TITLE	DESCRIPTION
ID3D12FunctionReflection	A function-reflection interface accesses function info.
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).
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.
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.
ID3D12LibraryReflection	A library-reflection interface accesses library info.
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.

TITLE	DESCRIPTION
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.
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.
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 serialized D3D12_ROOT_SIGNATURE_DESC data structure, of a serialized root signature version 1.0.
ID3D12ShaderReflection	A shader-reflection interface accesses shader information.
ID3D12ShaderReflectionConstantBuffer	This shader-reflection interface provides access to a constant buffer.
ID3D12ShaderReflectionType	This shader-reflection interface provides access to variable type.
ID3D12ShaderReflectionVariable	This shader-reflection interface provides access to a variable.
ID3D12SharingContract	Part of a contract between D3D11On12 diagnostic layers and graphics diagnostics tools.

TITLE	DESCRIPTION
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.
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.
IDMLBindingTable	Wraps a range of an application-managed descriptor heap, and is used by DirectML to create bindings for resources. To create this object, call IDMLDevice::CreateBindingTable.
IDMLCommandRecorder	Records dispatches of DirectML work into a Direct3D 12 command list.
IDMLCompiledOperator	Represents a compiled, efficient form of an operator suitable for execution on the GPU. To create this object, call IDMLDevice::CompileOperator.
IDMLDebugDevice	Controls the DirectML debug layers.
IDMLDevice	Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects.
IDMLDeviceChild	An interface implemented by all objects created from the DirectML device.
IDMLDispatchable	Implemented by objects that can be recorded into a command list for dispatch on the GPU, using IDMLCommandRecorder::RecordDispatch.
IDMLObject	An interface from which IDMLDevice and IDMLDeviceChild inherit directly (and all other interfaces, indirectly).
IDMLOperator	Represents a DirectML operator.
IDMLOperatorInitializer	Represents a specialized object whose purpose is to initialize compiled operators. To create an instance of this object, call IDMLDevice::CreateOperatorInitializer.
IDMLPageable	Implemented by objects that can be evicted from GPU memory, and hence that can be supplied to IDMLDevice::Evict and IDMLDevice::MakeResident.
IHolographicCameraInterop	Extends HolographicCamera to allow 2D texture resources to be created and used as back buffers for holographic rendering in Direct3D 12.

TITLE	DESCRIPTION
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

TITLE	DESCRIPTION
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_BLEND_DESC	Describes the blend state.
D3D12_BOX	Describes a 3D box.
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_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESCRIPTION	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.

TITLE	DESCRIPTION
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_DESC1	Describes depth-stencil state.
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.

TITLE	DESCRIPTION
D3D12_DISCARD_REGION	Describes details for the discard-resource operation.
D3D12_DISPATCH_ARGUMENTS	Describes dispatch parameters, for use by the compute shader.
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_AUTO_BREADCRUMBS_OUTPUT	Contains a pointer to the head of a linked list of D3D12_AUTO_BREADCRUMB_NODE objects.
D3D12_DRED_PAGE_FAULT_OUTPUT	Describes allocation data related to a GPU page fault on a given virtual address (VA).
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_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_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.

TITLE	DESCRIPTION
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_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_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_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_MODEL	Contains the supported shader model.
D3D12_FUNCTION_DESC	Describes a function.
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.

TITLE	DESCRIPTION
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_DESC	Allow or deny certain types of messages to pass through a filter.
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_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_META_COMMAND_DESC	Describes a meta command.
D3D12_META_COMMAND_PARAMETER_DESC	Describes a parameter to a meta command.
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_PARAMETER_DESC	Describes a function parameter.
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.

TITLE	DESCRIPTION
D3D12_PROTECTED_RESOURCE_SESSION_DESC	Describes flags 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_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_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 stucture 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_IN PUTS 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_IN PUTS structure to provide input data to a raytracing acceleration structure build operation.

TITLE	DESCRIPTION
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_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_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.

TITLE	DESCRIPTION
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 appear one after the other in a descriptor heap.
D3D12_ROOT_DESCRIPTOR_TABLE1	Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.
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_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_BYTECODE	Describes shader data.
D3D12_SHADER_DESC	Describes a shader.
D3D12_SHADER_INPUT_BIND_DESC	Describes how a shader resource is bound to a shader input.

TITLE	DESCRIPTION
D3D12_SHADER_RESOURCE_VIEW_DESC	Describes a shader-resource view.
D3D12_SHADER_TYPE_DESC	Describes a shader-variable type.
D3D12_SHADER_VARIABLE_DESC	Describes a shader variable.
D3D12_SIGNATURE_PARAMETER_DESC	Describes a shader signature.
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 with in 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_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_RANGE_UINT64	Describes a subresource memory range.
D3D12_SUBRESOURCE_TILING	Describes a tiled subresource volume.
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_DSV	Describes the subresource from a 1D texture that is accessible to a depth-stencil view.

TITLE	DESCRIPTION
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_UAV	Describes a unordered-access 1D texture resource.
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_SRV	Describes the subresources from an array of 2D textures to use in a shader-resource view.
D3D12_TEX2D_ARRAY_UAV	Describes an array of unordered-access 2D texture resources.
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_SRV	Describes the subresource from a 2D texture to use in a shader-resource view.
D3D12_TEX2D_UAV	Describes a unordered-access 2D texture resource.
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.

TITLE	DESCRIPTION
D3D12_TEX3D_UAV	Describes a unordered-access 3D texture resource.
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_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_SHAPE	Describes the shape of a tile by specifying its dimensions.
D3D12_TILED_RESOURCE_COORDINATE	Describes the coordinates of a tiled resource.
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.
DML_ACTIVATION_ELU_OPERATOR_DESC	Describes a DirectML operator that performs an exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } (\exp(x) - 1) * \alpha$.
DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC	Describes a DirectML activation operator that performs a hard sigmoid function on every element in the input, $f(x) = \max(0, \min(\alpha * x + \beta, 1))$.
DML_ACTIVATION_HARDMAX_OPERATOR_DESC	Describes a DirectML activation operator that performs a hardmax function on the input, $f(x) = \text{if } x_i == \max(x) \text{ then } 1 \text{ else } 0$ (but only for the first element in the axis).

TITLE	DESCRIPTION
DML_ACTIVATION_IDENTITY_OPERATOR_DESC	Describes a DirectML activation operator that performs the identity function $f(x) = x$. The operator effectively copies its input tensor to the output.
DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a leaky rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } x * \text{alpha}$.
DML_ACTIVATION_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs a linear activation function on every element in the input, $f(x) = \text{alpha} * x + \text{beta}$.
DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC	Describes a DirectML operator that performs a log-of-softmax activation function on the input, $f(x_i) = \log(\exp(x_i - \max(X)) / \sum(\exp(X - \max(X)))) = (x_i - \max(X)) - \log(\sum(\exp(x - \max(X))))$.
DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a parameterized rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else slope} * x$.
DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC	Describes a DirectML operator that performs a parametric softplus activation function on every element in the input, $f(x) = \text{alpha} * \text{log}(1 + \exp(\text{beta} * x))$.
DML_ACTIVATION_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \max(0, x)$.
DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC	Describes a DirectML operator that performs a scaled exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x > 0 \text{ then } \gamma * x \text{ else } \gamma * (\alpha * e^x - \alpha)$.
DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC	Describes a DirectML operator that performs a scaled hyperbolic tangent activation function on every element in the input, $f(x) = \text{alpha} * \tanh(\text{beta} * x)$.
DML_ACTIVATION_SHRINK_OPERATOR_DESC	Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ACTIVATION_SIGMOID_OPERATOR_DESC	Describes a DirectML operator that performs a sigmoid activation function on every element in the input, $f(x) = 1 / (1 + \exp(-x))$.
DML_ACTIVATION_SOFTMAX_OPERATOR_DESC	Describes a DirectML operator that performs a softmax activation function on the input, $f(x_i) = \exp(x_i - \max(X)) / \sum(\exp(X - \max(X))) = \exp(x_i - \max(X)) / \sum(\exp(x - \max(X)))$.
DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC	Describes a DirectML operator that performs a softplus activation function on every element in the input, $f(x) = \ln(1 + \exp(x))$.

TITLE	DESCRIPTION
DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC	Describes a DirectML operator that performs a softsign activation function on every element in the input, $f(x) = x / (1 + \text{abs}(x))$.
DML_ACTIVATION_TANH_OPERATOR_DESC	Describes a DirectML operator that performs a hyperbolic tangent activation function on every element in the input, $f(x) = (1 - \exp(-2 * x)) / (1 + \exp(-2 * x))$.
DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a thresholded rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x > \text{alpha} \text{ then } x \text{ else } 0$.
DML_AVERAGE_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs an average pooling function on the input, $y = (x_1 + x_2 + \dots) / \text{pool_size}$.
DML_BATCH_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a batch normalization function on the input, $y = \text{scale} * (x - \text{batchMean}) / \sqrt{\text{batchVariance} + \text{epsilon}} + \text{bias}$.
DML_BINDING_DESC	Contains the description of a binding so that you can add it to the binding table via a call to one of the IDMLBindingTable methods.
DML_BINDING_PROPERTIES	Contains information about the binding requirements of a particular compiled operator, or operator initializer. This struct is retrieved from IDMLDispatchable::GetBindingProperties.
DML_BINDING_TABLE_DESC	Specifies parameters to IDMLDevice::CreateBindingTable and IDMLBindingTable::Reset.
DML_BUFFER_ARRAY_BINDING	Specifies a resource binding that is an array of individual buffer bindings.
DML_BUFFER_BINDING	Specifies a resource binding described by a range of bytes in a Direct3D 12 buffer, represented by an offset and size into an ID3D12Resource.
DML_BUFFER_TENSOR_DESC	Describes a tensor that will be stored in a Direct3D 12 buffer resource.
DML_CAST_OPERATOR_DESC	Describes a DirectML data reorganization operator that performs the cast function $f(x) = \text{cast}(x)$, casting each element in the input to the data type of the output tensor, and storing the result in the corresponding element in the output.
DML_CONVOLUTION_OPERATOR_DESC	Describes a DirectML matrix multiplication operator that performs a convolution function on the input, $\text{out}[j] = x[i]*w[0] + x[i+1]*w[1] + x[i+2]*w[2] + \dots + x[i+k]*w[k] + \text{bias}$.
DML_DEPTH_TO_SPACE_OPERATOR_DESC	Describes a DirectML data reorganization operator that rearranges (permutes) data from depth into blocks of spatial data.

TITLE	DESCRIPTION
DML_DIAGONAL_MATRIX_OPERATOR_DESC	Describes a DirectML math operator that generates an identity-like matrix with ones on the major diagonal and zeros everywhere else.
DML_ELEMENT_WISE_ABS_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise absolute value function $f(x) = \text{abs}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The absolute value of x is its magnitude without regard to its sign.
DML_ELEMENT_WISE_ACOS_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arccosine function $f(x) = \text{acos}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic cosine function $f(x) = \log(x + \sqrt{x * x - 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ADD_OPERATOR_DESC	Describes a DirectML math operator that performs the function of adding every element in <i>ATensor</i> to its corresponding element in <i>BTensor</i> , $f(a, b) = a + b$.
DML_ELEMENT_WISE_ADD1_OPERATOR_DESC	Describes a DirectML math operator that performs the function of adding every element in <i>ATensor</i> to its corresponding element in <i>BTensor</i> , with the option for fused activation.
DML_ELEMENT_WISE_ASIN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arcsine function $f(x) = \text{asin}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ASINH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic sine function $f(x) = \log(x + \sqrt{x * x + 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATAN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arctangent function $f(x) = \text{atan}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATanh_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = (\log((1 + x) / (1 - x)) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_CEIL_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise ceiling function $f(x) = \text{ceil}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The ceiling of x is the smallest integer that is greater than or equal to x .
DML_ELEMENT_WISE_CLIP_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise clip function $f(x) = \text{clamp}(x * \text{scale} + \text{bias}, \text{minValue}, \text{maxValue})$, where the scale and bias terms are optional, and where $\text{clamp}(x) = \min(\maxValue, \max(\minValue, x))$.

TITLE	DESCRIPTION
DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC	Describes a DirectML operator that performs the element-wise constant power function $f(x) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional. The power function raises every element in the input to the power of the exponent.
DML_ELEMENT_WISE_COS_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise cosine function $f(x) = \cos(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_COSH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise hyperbolic cosine function $f(x) = ((e^x + e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs the linear dequantize function on every element in InputTensor with respect to its corresponding element in ScaleTensor and ZeroPointTensor, $f(\text{input}, \text{scale}, \text{zero_point}) = (\text{input} - \text{zero_point}) * \text{scale}$.
DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC	Describes a DirectML math operator that performs the function of dividing every element in ATensor by its corresponding element in BTensor, $f(a, b) = a / b$.
DML_ELEMENT_WISE_ERF_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_EXP_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise floor function $f(x) = \text{floor}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The floor of x is the largest integer that is less than or equal to x.
DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC	Describes a DirectML generic operator that performs the element-wise identity function $f(x) = x * \text{scale} + \text{bias}$. The operator effectively copies its input tensor to the output, while applying optional scale and bias terms.
DML_ELEMENT_WISE_IF_OPERATOR_DESC	Describes a DirectML math operator that essentially performs a ternary <code>if</code> statement.
DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC	Describes a DirectML math operator that determines, elementwise, whether the input is NaN.
DML_ELEMENT_WISE_LOG_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural logarithm function $f(x) = \log(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC	Describes a DirectML math operator that performs a logical AND function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a \&\& b$.

TITLE	DESCRIPTION
DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC	Describes a DirectML math operator that performs a logical equality function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a == b)$.
DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC	Describes a DirectML math operator that performs a logical greater-than function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a > b)$.
DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC	Describes a DirectML math operator that performs a logical less-than function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a < b)$.
DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC	Describes a DirectML math operator that performs a logical NOT function on every element in the input, $f(x) = !x$.
DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC	Describes a DirectML math operator that performs a logical OR function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a \parallel b$.
DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC	Describes a DirectML math operator that performs a logical exclusive OR (XOR) function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a \text{ xor } b$.
DML_ELEMENT_WISE_MAX_OPERATOR_DESC	Describes a DirectML math reduction operator that performs a maximum function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = \max(a, b)$.
DML_ELEMENT_WISE_MEAN_OPERATOR_DESC	Describes a DirectML math reduction operator that performs an arithmetic mean function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a + b) / 2$.
DML_ELEMENT_WISE_MIN_OPERATOR_DESC	Describes a DirectML math reduction operator that performs a minimum function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = \min(a, b)$.
DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC	Describes a DirectML math operator that performs the function of multiplying every element in ATensor by its corresponding element in BTensor, $f(a, b) = a * b$.
DML_ELEMENT_WISE_POW_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise power function $f(x, \text{exponent}) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs the linear quantize function on every element in InputTensor with respect to its corresponding element in ScaleTensor and ZeroPointTensor, $f(\text{input}, \text{scale}, \text{zero_point}) = \text{clamp}(\text{round}(\text{input} / \text{scale}) + \text{zero_point}, 0, 255)$.
DML_ELEMENT_WISE_RECIP_OPERATOR_DESC	Describes a DirectML math operator that performs a reciprocal function on every element in the input, $f(x) = 1 / (x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

TITLE	DESCRIPTION
DML_ELEMENT_WISE_SIGN_OPERATOR_DESC	Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ELEMENT_WISE_SIN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise sine function $f(x) = \sin(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SINH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise hyperbolic sine function $f(x) = ((e^x - e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SQRT_OPERATOR_DESC	Describes a DirectML math operator that performs a square root function on every element in the input, $f(x) = \sqrt{x * \text{scale} + \text{bias}}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC	Describes a DirectML math operator that performs the function of subtracting every element in BTensor from its corresponding element in ATensor, $f(a, b) = a - b$.
DML_ELEMENT_WISE_TAN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise tangent function $f(x) = \tan(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_TANH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = \tanh(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise threshold function $f(x) = \max(x * \text{scale} + \text{bias}, \text{min})$, where the scale and bias terms are optional. The threshold of x with respect to min is the larger of the two values.
DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT	Provides detail about whether a DirectML device supports a particular data type within tensors.
DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT	Used to query a DirectML device for its support for a particular data type within tensors.
DML_GATHER_OPERATOR_DESC	Describes a DirectML data reorganization operator which, when given a data tensor of rank $r \geq 1$, and an indices tensor of rank q, gathers the entries in the axis dimension of the data (by default, the outermost one is $\text{axis} == 0$) indexed by indices, and concatenates them in an output tensor of rank $q + (r - 1)$.
DML_GEMM_OPERATOR_DESC	Describes a DirectML operator that performs a general matrix multiplication function on the input, $y = \alpha * \text{transposeA}(A) * \text{transposeB}(B) + \beta * C$.
DML_GRU_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a (standard layers) one-layer gated recurrent unit (GRU) function on the input.

TITLE	DESCRIPTION
DML_JOIN_OPERATOR_DESC	Describes a DirectML operator that performs a join function on an array of input tensors.
DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a local response normalization (LRN) function on the input, $y = x / (\text{bias} + (\alpha / \text{size}) * \sum(x_i^2 \text{ for every } x_i \text{ in the local region}))^\beta$.
DML_LP_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs an Lp-normalization function along the specified axis of the input tensor.
DML_LP_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs an Lp pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = (x_1^p + x_2^p + \dots + x_n^p)^{(1/p)}$ where $X \rightarrow Y$ reduced for each kernel.
DML_LSTM_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a one-layer long short term memory (LSTM) function on the input.
DML_MAX_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = \max(x_1 + x_2 + \dots + x_{\text{pool_size}})$.
DML_MAX_POOLING1_OPERATOR_DESC	Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths).
DML_MAX_UNPOOLING_OPERATOR_DESC	Describes a DirectML operator that fills the output tensor of the given shape (either explicit, or the input shape plus padding) with zeros, then writes each value from the input tensor into the output tensor at the element offset from the corresponding indices array.
DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a mean variance normalization function on the input tensor.
DML_ONE_HOT_OPERATOR_DESC	Describes a DirectML operator that generates a tensor with each element filled with two values—either an 'on' or an 'off' value.
DML_OPERATOR_DESC	A generic container for an operator description. You construct DirectML operators using the parameters specified in this struct. See <code>IDMLDevice::CreateOperator</code> for additional details.
DML_PADDING_OPERATOR_DESC	Describes a DirectML data reorganization operator that inflates the input tensor with zeroes (or some other value) on the edges.
DML_REDUCE_OPERATOR_DESC	Describes a DirectML operator that performs the specified reduction function on the input.

TITLE	DESCRIPTION
DML_RESAMPLE_OPERATOR_DESC	Describes a DirectML operator that resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size.
DML_RNN_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a one-layer simple recurrent neural network (RNN) function on the input.
DML_ROI_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs a pooling function across the input tensor (according to regions of interest, or ROIs).
DML_SCALE_BIAS	Contains the values of scale and bias terms supplied to a DirectML operator. Scale and bias have the effect of applying the function $g(x) = x * \text{Scale} + \text{Bias}$.
DML_SCATTER_OPERATOR_DESC	Describes a DirectML operator that copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor.
DML_SIZE_2D	Contains values that can represent the size (as supplied to a DirectML operator) of a 2-D plane of elements within a tensor, or a 2-D scale, or any 2-D width/height value.
DML_SLICE_OPERATOR_DESC	Describes a DirectML data reorganization operator that produces a slice of the input tensor along multiple axes.
DML_SPACE_TO_DEPTH_OPERATOR_DESC	Describes a DirectML data reorganization operator that rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.
DML_SPLIT_OPERATOR_DESC	Describes a DirectML data reorganization operator that splits the input tensor into multiple output tensors, along the specified axis.
DML_TENSOR_DESC	A generic container for a DirectML tensor description.
DML_TILE_OPERATOR_DESC	Describes a DirectML data reorganization operator that constructs an output tensor by tiling the input tensor.
DML_TOP_K_OPERATOR_DESC	Describes a DirectML reduction operator that retrieves the top K elements along a specified axis.
DML_UPSAMPLE_2D_OPERATOR_DESC	Describes a DirectML imaging operator that upsamples the image contained in the input tensor. Each dimension value of the output tensor is $\text{output_dimension} = \text{floor}(\text{input_dimension} * \text{scale})$.
DML_VALUE_SCALE_2D_OPERATOR_DESC	Describes a DirectML operator that performs an element-wise scale-and-bias function on the values in the input tensor.

d3d11on12.h header

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

- [Direct3D 12 Graphics](#) d3d11on12.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
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

TITLE	DESCRIPTION
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

TITLE	DESCRIPTION
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.

D3D11_RESOURCE_FLAGS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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 QLs, 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

Header	d3d11on12.h

See also

[11on12 Structures](#)

D3D11On12CreateDevice function

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

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

Parameters

`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.

`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.

`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.

`ppDevice`

Type: `ID3D11Device**`

Pointer to the returned `ID3D11Device`. May be NULL.

`ppImmediateContext`

Type: `ID3D11DeviceContext**`

A pointer to the returned `ID3D11DeviceContext`. May be NULL.

`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.

Requirements

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

See also

[11on12 Functions](#)

ID3D11On12Device interface

5/13/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d11on12.h

See also

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

ID3D11On12Device::AcquireWrappedResources method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void AcquireWrappedResources(  
    ID3D11Resource * const *ppResources,  
    UINT             NumResources  
)
```

Parameters

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 [IDXGIFKeyedMutex::AcquireSync](#) instead.

Examples

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

```

// Render text over D3D12 using D2D via the 11On12 device.
void D3D1211on12::RenderUI()
{
    D2D1_SIZE_F rtSize = m_d2dRenderTarget->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_d3d11On12Device->AcquireWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(), 1);

    // Render text directly to the back buffer.
    m_d2dDeviceContext->SetTarget(m_d2dRenderTarget[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_d3d11On12Device->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

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

See also

[ID3D11On12Device](#)

ID3D11On12Device::CreateWrappedResource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

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

Parameters

`pResource12`

Type: `IUnknown*`

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

`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(ID3D12Resource)` will get the `GUID` of the interface to a wrapped resource.

`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

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

See also

[ID3D11On12Device](#)

ID3D11On12Device::ReleaseWrappedResources method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Releases D3D11 resources that were wrapped for D3D 11on12.

Syntax

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

Parameters

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 [IDXGIKeyedMutex::ReleaseSync](#) instead.

Examples

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

```

// Render text over D3D12 using D2D via the 11On12 device.
void D3D1211on12::RenderUI()
{
    D2D1_SIZE_F rtSize = m_d2dRenderTarget->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_d3d11On12Device->AcquireWrappedResources(m_wrappedBackBuffers[m_frameIndex].GetAddressOf(), 1);

    // Render text directly to the back buffer.
    m_d2dDeviceContext->SetTarget(m_d2dRenderTarget[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_d3d11On12Device->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

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

See also

[ID3D11On12Device](#)

d3d12.h header

2/7/2020 • 23 minutes to read • [Edit Online](#)

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

- [Direct3D 12 Graphics](#) d3d12.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
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.
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.

TITLE	DESCRIPTION
ID3D12Device6	Represents a virtual adapter. This interface extends ID3D12Device5 .
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.
ID3D12DeviceRemovedExtendedDataSettings	This interface controls Device Removed Extended Data (DRED) settings.
ID3D12Fence	Represents a fence, an object used for synchronization of the CPU and one or more GPUs.
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).
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.
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.

TITLE	DESCRIPTION
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.
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.
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 serialized D3D12_ROOT_SIGNATURE_DESC data structure, of a serialized root signature version 1.0.
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.

TITLE	DESCRIPTION
ID3D12StateObjectProperties	Provides methods for getting and setting the properties of an ID3D12StateObject.
ID3D12Tools	This interface is used to configure the runtime for tools such as PIX. It's 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.

Functions

TITLE	DESCRIPTION
D3D12CreateDevice	Creates a device that represents the display adapter.
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.
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.

Structures

TITLE	DESCRIPTION
D3D12_AUTO_BREADCRUMB_NODE	Represents Device Removed Extended Data (DRED) auto-breadcrumb data as a node in a linked list.
D3D12_BLEND_DESC	Describes the blend state.
D3D12_BOX	Describes a 3D box.
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 an unordered-access view.

TITLE	DESCRIPTION
D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESCRIPTOR	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_DESC1	Describes depth-stencil state.
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.

TITLE	DESCRIPTION
D3D12_DEVICE_REMOVED_EXTENDED_DATA1	Represents Device Removed Extended Data (DRED) version 1.1 data.
D3D12_DISCARD_REGION	Describes details for the discard-resource operation.
D3D12_DISPATCH_ARGUMENTS	Describes dispatch parameters, for use by the compute shader.
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_AUTO_BREADCRUMBS_OUTPUT	Contains a pointer to the head of a linked list of D3D12_AUTO_BREADCRUMB_NODE objects.
D3D12_DRED_PAGE_FAULT_OUTPUT	Describes allocation data related to a GPU page fault on a given virtual address (VA).
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.

TITLE	DESCRIPTION
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_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_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_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_MODEL	Contains the supported shader model.
D3D12_GLOBAL_ROOT_SIGNATURE	Defines a global root signature state subobject that will be used with associated shaders.

TITLE	DESCRIPTION
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_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_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_QUERY_DATA_PIPELINE_STATISTICS	Query information about graphics-pipeline activity in between calls to BeginQuery and EndQuery.

TITLE	DESCRIPTION
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_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 stucture 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_IN PUTS 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_IN PUTS 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.

TITLE	DESCRIPTION
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_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_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_TRANSITION_BARRIER	Describes the transition of subresources between different usages.

TITLE	DESCRIPTION
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 appear one after the other in a descriptor heap.
D3D12_ROOT_DESCRIPTOR_TABLE1	Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.
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_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_BYTCODE	Describes shader data.
D3D12_SHADER_RESOURCE_VIEW_DESC	Describes a shader-resource view.
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.

TITLE	DESCRIPTION
D3D12_STATE_SUBOBJECT	Represents a subobject with in 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_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_RANGE_UINT64	Describes a subresource memory range.
D3D12_SUBRESOURCE_TILING	Describes a tiled subresource volume.
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_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_UAV	Describes a unordered-access 1D texture resource.
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.

TITLE	DESCRIPTION
D3D12_TEX2D_ARRAY_SRV	Describes the subresources from an array of 2D textures to use in a shader-resource view.
D3D12_TEX2D_ARRAY_UAV	Describes an array of unordered-access 2D texture resources.
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_SRV	Describes the subresource from a 2D texture to use in a shader-resource view.
D3D12_TEX2D_UAV	Describes a unordered-access 2D texture resource.
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_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_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.

TITLE	DESCRIPTION
D3D12_TILE_SHAPE	Describes the shape of a tile by specifying its dimensions.
D3D12_TILED_RESOURCE_COORDINATE	Describes the coordinates of a tiled resource.
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

TITLE	DESCRIPTION
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_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_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.

TITLE	DESCRIPTION
D3D12_BUFFER_UAV_FLAGS	Identifies unordered-access view options for a buffer resource.
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	
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_QUEUE_FLAGS	Specifies flags to be used when creating a command queue.
D3D12_COMMAND_QUEUE_PRIORITY	Defines priority levels for a command queue.
D3D12_COMPARISON_FUNC	Specifies comparison options.
D3D12_CONSERVATIVE_RASTERIZATION_MODE	Identifies whether conservative rasterization is on or off.
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, D3D12DDI_HANDLETYPE enumeration values.
D3D12_DRED_ENABLEMENT	Defines constants that specify render/compute GPU operations.

TITLE	DESCRIPTION
D3D12_DRED_FLAGS	Defines constants used in the D3D12_DEVICE_REMOVED_EXTENDED_DATA structure to specify control flags for the Direct3D runtime.
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_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_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.

TITLE	DESCRIPTION
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_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_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_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.

TITLE	DESCRIPTION
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_IN_PUTS 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_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.

TITLE	DESCRIPTION
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_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_SERIALIZED_DATA_TYPE	Specifies the type of serialized data. Use a value from this enumeration when calling ID3D12Device5::CheckDriverMatchingIdentifier.
D3D12_SHADER_CACHE_SUPPORT_FLAGS	Describes the level of support for shader caching in the current graphics driver.
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.

TITLE	DESCRIPTION
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_COPY_TYPE	Specifies what type of texture copy is to take place.
D3D12_TEXTURE_LAYOUT	Specifies texture layout options.
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_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_WRITEBUFFERIMMEDIATE_MODE	Specifies the mode used by a WriteBufferImmediate operation.

D3D_ROOT_SIGNATURE_VERSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the version of root signature layout.

Syntax

```
typedef enum D3D_ROOT_SIGNATURE_VERSION {
    D3D_ROOT_SIGNATURE_VERSION_1,
    D3D_ROOT_SIGNATURE_VERSION_1_0,
    D3D_ROOT_SIGNATURE_VERSION_1_1
} ;
```

Constants

D3D_ROOT_SIGNATURE_VERSION_1	Version one of root signature layout.
D3D_ROOT_SIGNATURE_VERSION_1_0	Version one of root signature layout.
D3D_ROOT_SIGNATURE_VERSION_1_1	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

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

See also

[Core Enumerations](#)

D3D_SHADER_MODEL enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a shader model.

Syntax

```
typedef enum D3D_SHADER_MODEL {  
    D3D_SHADER_MODEL_5_1,  
    D3D_SHADER_MODEL_6_0,  
    D3D_SHADER_MODEL_6_1,  
    D3D_SHADER_MODEL_6_2,  
    D3D_SHADER_MODEL_6_3,  
    D3D_SHADER_MODEL_6_4,  
    D3D_SHADER_MODEL_6_5,  
    D3D_SHADER_MODEL_6_6  
} ;
```

Constants

D3D_SHADER_MODEL_5_1	Indicates shader model 5.1.
D3D_SHADER_MODEL_6_0	Indicates shader model 6.0.
D3D_SHADER_MODEL_6_1	Indicates shader model 6.1.
D3D_SHADER_MODEL_6_2	
D3D_SHADER_MODEL_6_3	
D3D_SHADER_MODEL_6_4	

Remarks

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

Requirements

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

See also

[Core Enumerations](#)

D3D12_AUTO_BREADCRUMB_NODE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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 index (within the array pointed to by `pCommandHistory`) of the last render/compute operation that was completed by the GPU while executing the associated command list.

`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

Header

`d3d12.h`

See also

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

D3D12_BACKGROUND_PROCESSING_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

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

Constants

D3D12_BACKGROUND_PROCESSING_MODE_ALLOWED	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

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

See also

[Core enumerations](#)

D3D12_BLEND enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_BLEND {
    D3D12_BLEND_ZERO,
    D3D12_BLEND_ONE,
    D3D12_BLEND_SRC_COLOR,
    D3D12_BLEND_INV_SRC_COLOR,
    D3D12_BLEND_SRC_ALPHA,
    D3D12_BLEND_INV_SRC_ALPHA,
    D3D12_BLEND_DEST_ALPHA,
    D3D12_BLEND_INV_DEST_ALPHA,
    D3D12_BLEND_DEST_COLOR,
    D3D12_BLEND_INV_DEST_COLOR,
    D3D12_BLEND_SRC_ALPHA_SAT,
    D3D12_BLEND_BLEND_FACTOR,
    D3D12_BLEND_INV_BLEND_FACTOR,
    D3D12_BLEND_SRC1_COLOR,
    D3D12_BLEND_INV_SRC1_COLOR,
    D3D12_BLEND_SRC1_ALPHA,
    D3D12_BLEND_INV_SRC1_ALPHA
} ;
```

Constants

D3D12_BLEND_ZERO	The blend factor is (0, 0, 0, 0). No pre-blend operation.
D3D12_BLEND_ONE	The blend factor is (1, 1, 1, 1). No pre-blend operation.
D3D12_BLEND_SRC_COLOR	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	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	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	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	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	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	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	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 $1 - \text{RGB}$.
D3D12_BLEND_SRC_ALPHA_SAT	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	The blend factor is the blend factor set with ID3D12GraphicsCommandList::OMSetBlendFactor . No pre-blend operation.
D3D12_BLEND_INV_BLEND_FACTOR	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	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	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	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	The blend factor is data sources as alpha data output by a pixel shader. The pre-blend operation inverts the data, generating $1 - A$. This blend factor supports dual-source color blending.

Remarks

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

Requirements

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

See also

[Core Enumerations](#)

D3D12_BLEND_DESC structure

4/28/2020 • 2 minutes to read • [Edit Online](#)

Describes the blend state.

Syntax

```
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

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.

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

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

See also

[Core structures](#)

D3D12_BLEND_OP enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies RGB or alpha blending operations.

Syntax

```
typedef enum D3D12_BLEND_OP {  
    D3D12_BLEND_OP_ADD,  
    D3D12_BLEND_OP_SUBTRACT,  
    D3D12_BLEND_OP_REV_SUBTRACT,  
    D3D12_BLEND_OP_MIN,  
    D3D12_BLEND_OP_MAX  
} ;
```

Constants

D3D12_BLEND_OP_ADD	Add source 1 and source 2.
D3D12_BLEND_OP_SUBTRACT	Subtract source 1 from source 2.
D3D12_BLEND_OP_REV_SUBTRACT	Subtract source 2 from source 1.
D3D12_BLEND_OP_MIN	Find the minimum of source 1 and source 2.
D3D12_BLEND_OP_MAX	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

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

See also

[Core Enumerations](#)

D3D12_BOX structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a 3D box.

Syntax

```
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 `top - bottom` equals the height of the box.

`back`

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

Remarks

This structure is used by the methods `WriteToSubresource`, `ReadFromSubresource` and `CopyTextureRegion`.

Requirements

Header	d3d12.h

See also

CD3DX12_BOX

Core Structures

D3D12_BUFFER_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

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

Members

FirstElement

Number of bytes 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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_BUFFER_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

`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.

Requirements

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

See also

[Core Structures](#)

D3D12_BUFFER_SRV_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies how to view a buffer resource.

Syntax

```
typedef enum D3D12_BUFFER_SRV_FLAGS {  
    D3D12_BUFFER_SRV_FLAG_NONE,  
    D3D12_BUFFER_SRV_FLAG_RAW  
} ;
```

Constants

D3D12_BUFFER_SRV_FLAG_NONE	Indicates a default view.
D3D12_BUFFER_SRV_FLAG_RAW	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_BUFFER_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header

d3d12.h

See also

[Core Structures](#)

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

D3D12_BUFFER_UAV_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies unordered-access view options for a buffer resource.

Syntax

```
typedef enum D3D12_BUFFER_UAV_FLAGS {  
    D3D12_BUFFER_UAV_FLAG_NONE,  
    D3D12_BUFFER_UAV_FLAG_RAW  
} ;
```

Constants

D3D12_BUFFER_UAV_FLAG_NONE	Indicates a default view.
D3D12_BUFFER_UAV_FLAG_RAW	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

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

See also

[Core Enumerations](#)

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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](#).

The memory pointed to 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.

The memory pointed to 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.

ScratchAccelerationStructureData

Requirements

Header	d3d12.h

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	
d3d12.h	

D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_TOOLS_VISUALIZATION_HEADER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the GPU memory layout of an acceleration structure visualization.

Syntax

```
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

Header

d3d12.h

D3D12_CACHED_PIPELINE_STATE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Stores a pipeline state.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_CLEAR_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies what to clear from the depth stencil view.

Syntax

```
typedef enum D3D12_CLEAR_FLAGS {  
    D3D12_CLEAR_FLAG_DEPTH,  
    D3D12_CLEAR_FLAG_STENCIL  
} ;
```

Constants

D3D12_CLEAR_FLAG_DEPTH	Indicates the depth buffer should be cleared.
D3D12_CLEAR_FLAG_STENCIL	Indicates the stencil buffer should be cleared.

Remarks

This enum is used by [ID3D12GraphicsCommandList::ClearDepthStencilView](#). The flags can be combined to clear all.

Requirements

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

See also

[Core Enumerations](#)

D3D12_CLEAR_VALUE structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

Header	d3d12.h

See also

[CD3DX12_CLEAR_VALUE](#)

[Core Structures](#)

D3D12_COLOR_WRITE_ENABLE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_COLOR_WRITE_ENABLE {
    D3D12_COLOR_WRITE_ENABLE_RED,
    D3D12_COLOR_WRITE_ENABLE_GREEN,
    D3D12_COLOR_WRITE_ENABLE_BLUE,
    D3D12_COLOR_WRITE_ENABLE_ALPHA,
    D3D12_COLOR_WRITE_ENABLE_ALL
} ;
```

Constants

D3D12_COLOR_WRITE_ENABLE_RED	Allow data to be stored in the red component.
D3D12_COLOR_WRITE_ENABLE_GREEN	Allow data to be stored in the green component.
D3D12_COLOR_WRITE_ENABLE_BLUE	Allow data to be stored in the blue component.
D3D12_COLOR_WRITE_ENABLE_ALPHA	Allow data to be stored in the alpha component.
D3D12_COLOR_WRITE_ENABLE_ALL	Allow data to be stored in all components.

Remarks

This enum is used by the [D3D12_RENDER_TARGET_BLEND_DESC](#) structure.

Requirements

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

See also

[CD3DX12_BLEND_DESC](#)

[Core Enumerations](#)

D3D12_COMMAND_LIST_SUPPORT_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Used to determine which kinds of command lists are capable of supporting various operations. For example, whether a command list supports immediate writes.

Syntax

```
typedef enum D3D12_COMMAND_LIST_SUPPORT_FLAGS {  
    D3D12_COMMAND_LIST_SUPPORT_FLAG_NONE,  
    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

D3D12_COMMAND_LIST_SUPPORT_FLAG_NONE	Specifies that no command list supports the operation in question.
D3D12_COMMAND_LIST_SUPPORT_FLAG_DIRECT	Specifies that direct command lists can support the operation in question.
D3D12_COMMAND_LIST_SUPPORT_FLAG_BUNDLE	Specifies that command list bundles can support the operation in question.
D3D12_COMMAND_LIST_SUPPORT_FLAG_COMPUTE	Specifies that compute command lists can support the operation in question.
D3D12_COMMAND_LIST_SUPPORT_FLAG_COPY	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

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

See also

[Core Enumerations](#)

[D3D12_COMMAND_LIST_TYPE](#).

D3D12_COMMAND_LIST_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of a command list.

Syntax

```
typedef enum D3D12_COMMAND_LIST_TYPE {
    D3D12_COMMAND_LIST_TYPE_DIRECT,
    D3D12_COMMAND_LIST_TYPE_BUNDLE,
    D3D12_COMMAND_LIST_TYPE_COMPUTE,
    D3D12_COMMAND_LIST_TYPE_COPY,
    D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE,
    D3D12_COMMAND_LIST_TYPE_VIDEO_PROCESS,
    D3D12_COMMAND_LIST_TYPE_VIDEO_ENCODE
} ;
```

Constants

D3D12_COMMAND_LIST_TYPE_DIRECT	Specifies a command buffer that the GPU can execute. A direct command list doesn't inherit any GPU state.
D3D12_COMMAND_LIST_TYPE_BUNDLE	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	Specifies a command buffer for computing.
D3D12_COMMAND_LIST_TYPE_COPY	Specifies a command buffer for copying.
D3D12_COMMAND_LIST_TYPE_VIDEO_DECODE	Specifies a command buffer for video decoding.
D3D12_COMMAND_LIST_TYPE_VIDEO_PROCESS	Specifies a command buffer for video processing.

Remarks

This enum is used by the following methods:

- [CreateCommandAllocator](#)
- [CreateCommandQueue](#)
- [CreateCommandList](#)

Requirements

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

See also

[Core Enumerations](#)

D3D12_COMMAND_QUEUE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a command queue.

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_COMMAND_QUEUE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies flags to be used when creating a command queue.

Syntax

```
typedef enum D3D12_COMMAND_QUEUE_FLAGS {  
    D3D12_COMMAND_QUEUE_FLAG_NONE,  
    D3D12_COMMAND_QUEUE_FLAG_DISABLE_GPU_TIMEOUT  
} ;
```

Constants

D3D12_COMMAND_QUEUE_FLAG_NONE	Indicates a default command queue.
D3D12_COMMAND_QUEUE_FLAG_DISABLE_GPU_TIMEOUT	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_COMMAND_QUEUE_PRIORITY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines priority levels for a command queue.

Syntax

```
typedef enum D3D12_COMMAND_QUEUE_PRIORITY {
    D3D12_COMMAND_QUEUE_PRIORITY_NORMAL,
    D3D12_COMMAND_QUEUE_PRIORITY_HIGH,
    D3D12_COMMAND_QUEUE_PRIORITY_GLOBAL_REALTIME
} ;
```

Constants

D3D12_COMMAND_QUEUE_PRIORITY_NORMAL	Normal priority.
D3D12_COMMAND_QUEUE_PRIORITY_HIGH	High priority.
D3D12_COMMAND_QUEUE_PRIORITY_GLOBAL_REALTIME	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

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

See also

[Core Enumerations](#)

D3D12_COMMAND_SIGNATURE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the arguments (parameters) of a command signature.

Syntax

```
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 argument of a command signature, 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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_COMPARISON_FUNC enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies comparison options.

Syntax

```
typedef enum D3D12_COMPARISON_FUNC {  
    D3D12_COMPARISON_FUNC_NEVER,  
    D3D12_COMPARISON_FUNC_LESS,  
    D3D12_COMPARISON_FUNC_EQUAL,  
    D3D12_COMPARISON_FUNC_LESS_EQUAL,  
    D3D12_COMPARISON_FUNC_GREATER,  
    D3D12_COMPARISON_FUNC_NOT_EQUAL,  
    D3D12_COMPARISON_FUNC_GREATER_EQUAL,  
    D3D12_COMPARISON_FUNC_ALWAYS  
} ;
```

Constants

D3D12_COMPARISON_FUNC_NEVER	Never pass the comparison.
D3D12_COMPARISON_FUNC_LESS	If the source data is less than the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_EQUAL	If the source data is equal to the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_LESS_EQUAL	If the source data is less than or equal to the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_GREATER	If the source data is greater than the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_NOT_EQUAL	If the source data is not equal to the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_GREATER_EQUAL	If the source data is greater than or equal to the destination data, the comparison passes.
D3D12_COMPARISON_FUNC_ALWAYS	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

Header	
	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core Enumerations](#)

D3D12_COMPUTE_PIPELINE_STATE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a compute pipeline state object.

Syntax

```
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.

Flags

A [D3D12_PIPELINE_STATE_FLAGS](#) enumeration constant such as for "tool debug".

Remarks

This structure is used by [CreateComputePipelineState](#).

Requirements

Header

d3d12.h

See also

[Core Structures](#)

D3D12_CONSERVATIVE_RASTERIZATION_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies whether conservative rasterization is on or off.

Syntax

```
typedef enum D3D12_CONSERVATIVE_RASTERIZATION_MODE {  
    D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF,  
    D3D12_CONSERVATIVE_RASTERIZATION_MODE_ON  
} ;
```

Constants

D3D12_CONSERVATIVE_RASTERIZATION_MODE_OFF	Conservative rasterization is off.
D3D12_CONSERVATIVE_RASTERIZATION_MODE_ON	Conservative rasterization is on.

Remarks

This enum is used by the [D3D12_RASTERIZER_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[Conservative Rasterization](#)

[Core Enumerations](#)

[D3D12_CONSERVATIVE_RASTERIZATION_TIER](#)

D3D12_CONSERVATIVE_RASTERIZATION_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the tier level of conservative rasterization.

Syntax

```
typedef enum D3D12_CONSERVATIVE_RASTERIZATION_TIER {  
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_NOT_SUPPORTED,  
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_1,  
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_2,  
    D3D12_CONSERVATIVE_RASTERIZATION_TIER_3  
};
```

Constants

D3D12_CONSERVATIVE_RASTERIZATION_TIER_NOT_SUPPORTED	Conservative rasterization is not supported.
D3D12_CONSERVATIVE_RASTERIZATION_TIER_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.
D3D12_CONSERVATIVE_RASTERIZATION_TIER_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).
D3D12_CONSERVATIVE_RASTERIZATION_TIER_3	Tier 3 maintains a maximum 1/256 uncertainty region and adds support for inner input coverage. Inner input coverage adds the new value <code>SV_InnerCoverage</code> 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

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

See also

[Conservative Rasterization](#)

[Core Enumerations](#)

[D3D12_CONSERVATIVE_RASTERIZATION_MODE](#)

D3D12_CONSTANT_BUFFER_VIEW_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a constant buffer to view.

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_CPU_DESCRIPTOR_HANDLE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a CPU descriptor handle.

Syntax

```
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](#)

Requirements

Header	d3d12.h

See also

[CD3DX12_CPU_DESCRIPTOR_HANDLE](#)

Core Structures

D3D12_CPU_PAGE_PROPERTY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the CPU-page properties for the heap.

Syntax

```
typedef enum D3D12_CPU_PAGE_PROPERTY {  
    D3D12_CPU_PAGE_PROPERTY_UNKNOWN,  
    D3D12_CPU_PAGE_PROPERTY_NOT_AVAILABLE,  
    D3D12_CPU_PAGE_PROPERTY_WRITE_COMBINE,  
    D3D12_CPU_PAGE_PROPERTY_WRITE_BACK  
} ;
```

Constants

D3D12_CPU_PAGE_PROPERTY_UNKNOWN	The CPU-page property is unknown.
D3D12_CPU_PAGE_PROPERTY_NOT_AVAILABLE	The CPU cannot access the heap, therefore no page properties are available.
D3D12_CPU_PAGE_PROPERTY_WRITE_COMBINE	The CPU-page property is write-combined.
D3D12_CPU_PAGE_PROPERTY_WRITE_BACK	The CPU-page property is write-back.

Remarks

This enum is used by the [D3D12_HEAP_PROPERTIES](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_CROSS_NODE_SHARING_TIER enumeration

5/5/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_CROSS_NODE_SHARING_TIER {  
    D3D12_CROSS_NODE_SHARING_TIER_NOT_SUPPORTED,  
    D3D12_CROSS_NODE_SHARING_TIER_1_EMULATED,  
    D3D12_CROSS_NODE_SHARING_TIER_1,  
    D3D12_CROSS_NODE_SHARING_TIER_2,  
    D3D12_CROSS_NODE_SHARING_TIER_3  
} ;
```

Constants

D3D12_CROSS_NODE_SHARING_TIER_NOT_SUPPORTED	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	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	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: <ul style="list-style-type: none">• 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	<p>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:</p> <ul style="list-style-type: none"> • 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. <p>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.</p>
D3D12_CROSS_NODE_SHARING_TIER_3	<p>Indicates support for D3D12_HEAP_FLAG_ALLOW_SHADER_ATOMICS on heaps that are visible to multiple nodes.</p>

Remarks

This enum is used by the **CrossNodeSharingTier** member of the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_CULL_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies triangles facing a particular direction are not drawn.

Syntax

```
typedef enum D3D12_CULL_MODE {  
    D3D12_CULL_MODE_NONE,  
    D3D12_CULL_MODE_FRONT,  
    D3D12_CULL_MODE_BACK  
} ;
```

Constants

D3D12_CULL_MODE_NONE	Always draw all triangles.
D3D12_CULL_MODE_FRONT	Do not draw triangles that are front-facing.
D3D12_CULL_MODE_BACK	Do not draw triangles that are back-facing.

Remarks

Cull mode is specified in a [D3D12_RASTERIZER_DESC](#) structure.

Requirements

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

See also

[CD3DX12_RASTERIZER_DESC](#)

[Core Enumerations](#)

D3D12_DEPTH_STENCIL_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes depth-stencil state.

Syntax

```
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.

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 BackFace.StencilFunc	D3D12_COMPARISON_ALWAYS
FrontFace.StencilDepthFailOp and BackFace.StencilDepthFailOp	D3D12_STENCIL_OP_KEEP
FrontFace.StencilPassOp and BackFace.StencilPassOp	D3D12_STENCIL_OP_KEEP
FrontFace.StencilFailOp and BackFace.StencilFailOp	D3D12_STENCIL_OP_KEEP

The formats that support stenciling are DXGI_FORMAT_D24_UNORM_S8_UINT and DXGI_FORMAT_D32_FLOAT_S8X24_UINT.

Requirements

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

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

Core Structures

D3D12_DEPTH_STENCIL_DESC1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes depth-stencil state.

Syntax

```
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.

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 BackFace.StencilFunc	D3D12_COMPARISON_ALWAYS
FrontFace.StencilDepthFailOp and BackFace.StencilDepthFailOp	D3D12_STENCIL_OP_KEEP
FrontFace.StencilPassOp and BackFace.StencilPassOp	D3D12_STENCIL_OP_KEEP
FrontFace.StencilFailOp and BackFace.StencilFailOp	D3D12_STENCIL_OP_KEEP
DepthBoundsTestEnable	FALSE

The formats that support stenciling are DXGI_FORMAT_D24_UNORM_S8_UINT and DXGI_FORMAT_D32_FLOAT_S8X24_UINT.

Requirements

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

See also

[Core Structures](#)

D3D12_DEPTH_STENCIL_VALUE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a depth and stencil value.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_DEPTH_STENCIL_VIEW_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_DEPTH_STENCILOP_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header

d3d12.h

See also

[Core Structures](#)

D3D12_DEPTH_WRITE_MASK enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_DEPTH_WRITE_MASK {  
    D3D12_DEPTH_WRITE_MASK_ZERO,  
    D3D12_DEPTH_WRITE_MASK_ALL  
} ;
```

Constants

D3D12_DEPTH_WRITE_MASK_ZERO	Turn off writes to the depth-stencil buffer.
D3D12_DEPTH_WRITE_MASK_ALL	Turn on writes to the depth-stencil buffer.

Remarks

This enum is used by the [D3D12_DEPTH_STENCIL_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core Enumerations](#)

D3D12_DESCRIPTOR_HEAP_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the descriptor heap.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

D3D12_DESCRIPTOR_HEAP_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies options for a heap.

Syntax

```
typedef enum D3D12_DESCRIPTOR_HEAP_FLAGS {  
    D3D12_DESCRIPTOR_HEAP_FLAG_NONE,  
    D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE  
} ;
```

Constants

D3D12_DESCRIPTOR_HEAP_FLAG_NONE	Indicates default usage of a heap.
D3D12_DESCRIPTOR_HEAP_FLAG_SHADER_VISIBLE	<p>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 <i>without</i> 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.</p> <p>This flag only applies to CBV, SRV, UAV and samplers. It does not apply to other descriptor heap types since shaders do not directly reference the other types.</p>

Remarks

This enum is used by the [D3D12_DESCRIPTOR_HEAP_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[Core Enumerations](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

D3D12_DESCRIPTOR_HEAP_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a type of descriptor heap.

Syntax

```
typedef enum D3D12_DESCRIPTOR_HEAP_TYPE {
    D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV,
    D3D12_DESCRIPTOR_HEAP_TYPE_SAMPLER,
    D3D12_DESCRIPTOR_HEAP_TYPE_RTV,
    D3D12_DESCRIPTOR_HEAP_TYPE_DSV,
    D3D12_DESCRIPTOR_HEAP_TYPE_NUM_TYPES
} ;
```

Constants

D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV	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

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

See also

[Core Enumerations](#)

[Creating Descriptor Heaps](#)

Descriptor Heaps

D3D12_DESCRIPTOR_RANGE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a descriptor range.

Syntax

```
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

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

See also

[CD3DX12_DESCRIPTOR_RANGE](#)

[Core Structures](#)

D3D12_DESCRIPTOR_RANGE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_DESCRIPTOR_RANGE_FLAGS {
    D3D12_DESCRIPTOR_RANGE_FLAG_NONE,
    D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_VOLATILE,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_VOLATILE,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE,
    D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC,
    D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTOR_STATIC_KEEPING_BUFFER_BOUNDS_CHECKS
} ;
```

Constants

D3D12_DESCRIPTOR_RANGE_FLAG_NONE	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	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	Descriptors are static and the data is volatile.
D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE	Descriptors are static and data is static while set at execute.
D3D12_DESCRIPTOR_RANGE_FLAG_DATA_STATIC	Both descriptors and data are static. This maximizes the potential for driver optimization.

D3D12_DESCRIPTOR_RANGE_FLAG_DESCRIPTORS_STATIC_KEEPING_BUFFER_BOUNDS_CHECKS	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

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

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

D3D12_DESCRIPTOR_RANGE_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_DESCRIPTOR_RANGE_TYPE {  
    D3D12_DESCRIPTOR_RANGE_TYPE_SRV,  
    D3D12_DESCRIPTOR_RANGE_TYPE_UAV,  
    D3D12_DESCRIPTOR_RANGE_TYPE_CBV,  
    D3D12_DESCRIPTOR_RANGE_TYPE_SAMPLER  
} ;
```

Constants

D3D12_DESCRIPTOR_RANGE_TYPE_SRV	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

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

See also

[Core Enumerations](#)

D3D12_DESCRIPTOR_RANGE1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	
	d3d12.h

See also

[Core Structures](#)

[Root Signature Version 1.1](#)

D3D12_DEVICE_REMOVED_EXTENDED_DATA structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

See also

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

D3D12_DEVICE_REMOVED_EXTENDED_DATA1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header

d3d12.h

See also

- [Core structures](#)
- [D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA](#)
- [Use DRED to diagnose GPU faults](#)

D3D12_DISCARD_REGION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes details for the discard-resource operation.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_DISPATCH_ARGUMENTS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes dispatch parameters, for use by the compute shader.

Syntax

```
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:

```
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

Header	File
<code>d3d12.h</code>	

See also

Core Structures

D3D12_DISPATCH_RAYS_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the properties of a ray dispatch operation initiated with a call to [ID3D12GraphicsCommandList4::DispatchRays](#).

Syntax

```
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

Header	
	d3d12.h

D3D12_DRAW_ARGUMENTS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes parameters for drawing instances.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_DRAW_INDEXED_ARGUMENTS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes parameters for drawing indexed instances.

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

D3D12_DRED_ALLOCATION_NODE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

See also

- Core structures
- Use DRED to diagnose GPU faults

D3D12_DRED_AUTO_BREADCRUMBS_OUTPUT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	d3d12.h

See also

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

D3D12_DRED_PAGE_FAULT_OUTPUT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	d3d12.h

See also

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

D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the result of a call to [ID3D12Device5::CheckDriverMatchingIdentifier](#) which queries whether serialized data is compatible with the current device and driver version.

Syntax

```
typedef enum D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS {
    D3D12_DRIVER_MATCHING_IDENTIFIER_COMPATIBLE_WITH_DEVICE,
    D3D12_DRIVER_MATCHING_IDENTIFIER_UNSUPPORTED_TYPE,
    D3D12_DRIVER_MATCHING_IDENTIFIER_UNRECOGNIZED,
    D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_VERSION,
    D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_TYPE
} ;
```

Constants

D3D12_DRIVER_MATCHING_IDENTIFIER_COMPATIBLE_WITH_DEVICE	Serialized data is compatible with the current device/driver.
D3D12_DRIVER_MATCHING_IDENTIFIER_UNSUPPORTED_TYPE	The specified D3D12_SERIALIZED_DATA_TYPE specified is unknown or unsupported.
D3D12_DRIVER_MATCHING_IDENTIFIER_UNRECOGNIZED	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.
D3D12_DRIVER_MATCHING_IDENTIFIER_INCOMPATIBLE_VERSION	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	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

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

D3D12_DSV_DIMENSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies how to access a resource used in a depth-stencil view.

Syntax

```
typedef enum D3D12_DSV_DIMENSION {  
    D3D12_DSV_DIMENSION_UNKNOWN,  
    D3D12_DSV_DIMENSION_TEXTURE1D,  
    D3D12_DSV_DIMENSION_TEXTURE1DARRAY,  
    D3D12_DSV_DIMENSION_TEXTURE2D,  
    D3D12_DSV_DIMENSION_TEXTURE2DARRAY,  
    D3D12_DSV_DIMENSION_TEXTURE2DMS,  
    D3D12_DSV_DIMENSION_TEXTURE2DMSARRAY  
} ;
```

Constants

D3D12_DSV_DIMENSION_UNKNOWN	D3D12_DSV_DIMENSION_UNKNOWN is not a valid value for D3D12_DEPTH_STENCIL_VIEW_DESC and is not used.
D3D12_DSV_DIMENSION_TEXTURE1D	The resource will be accessed as a 1D texture.
D3D12_DSV_DIMENSION_TEXTURE1DARRAY	The resource will be accessed as an array of 1D textures.
D3D12_DSV_DIMENSION_TEXTURE2D	The resource will be accessed as a 2D texture.
D3D12_DSV_DIMENSION_TEXTURE2DARRAY	The resource will be accessed as an array of 2D textures.
D3D12_DSV_DIMENSION_TEXTURE2DMS	The resource will be accessed as a 2D texture with multi sampling.
D3D12_DSV_DIMENSION_TEXTURE2DMSARRAY	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

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

See also

[Core Enumerations](#)

D3D12_DSV_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies depth-stencil view options.

Syntax

```
typedef enum D3D12_DSV_FLAGS {  
    D3D12_DSV_FLAG_NONE,  
    D3D12_DSV_FLAG_READ_ONLY_DEPTH,  
    D3D12_DSV_FLAG_READ_ONLY_STENCIL  
} ;
```

Constants

D3D12_DSV_FLAG_NONE	Indicates a default view.
D3D12_DSV_FLAG_READ_ONLY_DEPTH	Indicates that depth values are read only.
D3D12_DSV_FLAG_READ_ONLY_STENCIL	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

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

See also

[Core Enumerations](#)

D3D12_DXIL_LIBRARY_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DXIL library state subobject that can be included in a state object.

Syntax

```
typedef struct D3D12_DXIL_LIBRARY_DESC {  
    D3D12_SHADER_BYTECODE DXILLibrary;  
    UINT                 NumExports;  
    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

Header	
d3d12.h	

D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

This subobject is unsupported in the current release.

Syntax

```
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.

pExports

Requirements

Header	d3d12.h

D3D12_ELEMENTS_LAYOUT enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes how the locations of elements are identified.

Syntax

```
typedef enum D3D12_ELEMENTS_LAYOUT {
    D3D12_ELEMENTS_LAYOUT_ARRAY,
    D3D12_ELEMENTS_LAYOUT_ARRAY_OF_POINTERS
} ;
```

Constants

D3D12_ELEMENTS_LAYOUT_ARRAY	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	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

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

D3D12_EXISTING_COLLECTION_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A state subobject describing an existing collection that can be included in a state object.

Syntax

```
typedef struct D3D12_EXISTING_COLLECTION_DESC {  
    ID3D12StateObject *pExistingCollection;  
    UINT              NumExports;  
    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

Header	d3d12.h

D3D12_EXPORT_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes an export from a state subobject such as a DXIL library or a collection state object.

Syntax

```
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

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

D3D12_EXPORT_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

The flags to apply when exporting symbols from a state subobject.

Syntax

```
typedef enum D3D12_EXPORT_FLAGS {
    D3D12_EXPORT_FLAG_NONE
} ;
```

Constants

D3D12_EXPORT_FLAG_NONE	No export flags.

Remarks

No export flags are defined in the current release.

Requirements

Header	d3d12.h

D3D12_FEATURE enumeration

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_FEATURE {
    D3D12_FEATURE_D3D12_OPTIONS,
    D3D12_FEATURE_ARCHITECTURE,
    D3D12_FEATURE_FEATURE_LEVELS,
    D3D12_FEATURE_FORMAT_SUPPORT,
    D3D12_FEATURE_MULTISAMPLE_QUALITY_LEVELS,
    D3D12_FEATURE_FORMAT_INFO,
    D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT,
    D3D12_FEATURE_SHADER_MODEL,
    D3D12_FEATURE_D3D12_OPTIONS1,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_SUPPORT,
    D3D12_FEATURE_ROOT_SIGNATURE,
    D3D12_FEATURE_ARCHITECTURE1,
    D3D12_FEATURE_D3D12_OPTIONS2,
    D3D12_FEATURE_SHADER_CACHE,
    D3D12_FEATURE_COMMAND_QUEUE_PRIORITY,
    D3D12_FEATURE_D3D12_OPTIONS3,
    D3D12_FEATURE_EXISTING_HEAPS,
    D3D12_FEATURE_D3D12_OPTIONS4,
    D3D12_FEATURE_SERIALIZATION,
    D3D12_FEATURE_CROSS_NODE,
    D3D12_FEATURE_D3D12_OPTIONS5,
    D3D12_FEATURE_D3D12_OPTIONS6,
    D3D12_FEATURE_QUERY_META_COMMAND,
    D3D12_FEATURE_D3D12_OPTIONS7,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPE_COUNT,
    D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_TYPES
} ;
```

Constants

D3D12_FEATURE_D3D12_OPTIONS	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	<p>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.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>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.</p> </div>
D3D12_FEATURE FEATURE_LEVELS	<p>Indicates a query for info about the feature levels supported. The corresponding data structure for this value is D3D12_FEATURE_DATA_FEATURE_LEVELS.</p>
D3D12_FEATURE_FORMAT_SUPPORT	<p>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.</p>
D3D12_FEATURE_MULTISAMPLE_QUALITY_LEVELS	<p>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.</p>
D3D12_FEATURE_FORMAT_INFO	<p>Indicates a query for the DXGI data format. The corresponding data structure for this value is D3D12_FEATURE_DATA_FORMAT_INFO.</p>
D3D12_FEATURE_GPU_VIRTUAL_ADDRESS_SUPPORT	<p>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.</p>
D3D12_FEATURE_SHADER_MODEL	<p>Indicates a query for the supported shader model. The corresponding data structure for this value is D3D12_FEATURE_DATA_SHADER_MODEL.</p>
D3D12_FEATURE_D3D12_OPTIONS1	<p>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.</p>
D3D12_FEATURE_PROTECTED_RESOURCE_SESSION_SUPPORT	<p>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.</p>
D3D12_FEATURE_ROOT_SIGNATURE	<p>Indicates a query for root signature version support. The corresponding data structure for this value is D3D12_FEATURE_DATA_ROOT_SIGNATURE.</p>

D3D12_FEATURE_ARCHITECTURE1	<p>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.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>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.</p> </div>
D3D12_FEATURE_D3D12_OPTIONS2	<p>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.</p>
D3D12_FEATURE_SHADER_CACHE	<p>Indicates a query for the level of support for shader caching. The corresponding data structure for this value is D3D12_FEATURE_DATA_SHADER_CACHE.</p>
D3D12_FEATURE_COMMAND_QUEUE_PRIORITY	<p>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.</p>
D3D12_FEATURE_D3D12_OPTIONS3	<p>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.</p>
D3D12_FEATURE_EXISTING_HEAPS	<p>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.</p>
D3D12_FEATURE_D3D12_OPTIONS4	<p>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.</p>
D3D12_FEATURE_SERIALIZATION	<p>Indicates a query for the level of support for heap serialization. The corresponding data structure for this value is D3D12_FEATURE_DATA_SERIALIZATION.</p>
D3D12_FEATURE_CROSS_NODE	<p>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.</p>
D3D12_FEATURE_D3D12_OPTIONS5	<p>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.</p>

D3D12_FEATURE_D3D12_OPTIONS6	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. For more info, see Variable-rate shading (VRS) , and the Direct3D 12 background processing spec .
D3D12_FEATURE_QUERY_META_COMMAND	Indicates a query for the level of support for metacommands. The corresponding data structure for this value is D3D12_FEATURE_DATA_QUERY_META_COMMAND .

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

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

See also

[Core enumerations](#)

[ID3D12Device::CheckFeatureSupport](#)

D3D12_FEATURE_DATA_ARCHITECTURE structure

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

```
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

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

See also

[Core Structures](#)

D3D12_FEATURE

D3D12_FEATURE_DATA_ARCHITECTURE1 structure

5/27/2020 • 4 minutes to read • [Edit Online](#)

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

```
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 [IDXGIAAdapter3::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

Header	
	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_COMMAND_QUEUE_PRIORITY structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_FEATURE

D3D12_FEATURE_DATA_D3D12_OPTIONS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes Direct3D 12 feature options in the current graphics driver.

Syntax

```
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

Header	
	d3d12.h

See also

[Conservative Rasterization](#)

[Core Structures](#)

[D3D12_FEATURE](#)

[Rasterizer Ordered Views](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the level of support for HLSL 6.0 wave operations.

Syntax

```
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. This capability is reserved for future expansion, and is not expected to be used by current applications.

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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS2 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS3 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS4 structure

6/19/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	File
	d3d12.h

D3D12_FEATURE_DATA_D3D12_OPTIONS5 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

RaytracingTier

Specifies the level of ray tracing support on the graphics device.

RaytracingTier

Remarks

Pass [D3D12_FEATURE_D3D12_OPTIONS5](#) to [ID3D12Device::CheckFeatureSupport](#) to retrieve a [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) structure.

Requirements

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_D3D12_OPTIONS6 structure

6/19/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

See also

[Variable-rate shading \(VRS\)](#), [Direct3D 12 background processing spec](#)

D3D12_FEATURE_DATA_EXISTING_HEAPS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[ID3D12Device::CreateCommittedResource](#)

[ID3D12Device::CreateHeap](#)

D3D12_FEATURE_DATA FEATURE_LEVELS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes info about the [feature levels](#) supported by the current graphics driver.

Syntax

```
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_LEVEL](#)s 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

Header		d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_FORMAT_INFO structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DXGI data format and plane count.

Syntax

```
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

```
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

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

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_FORMAT_SUPPORT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes which resources are supported by the current graphics driver for a given format.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[Typed unordered access view loads](#)

D3D12_FEATURE_DATA_GPU_VIRTUAL_ADDRESS_SUPPORT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Details the adapter's GPU virtual address space limitations, including maximum address bits per resource and per process.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_MULTISAMPLE_QUALITY_LEVELS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the multi-sampling image quality levels for a given format and sample count.

Syntax

```
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

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

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_PROTECTED_RESOURCE_SESSION_SUPPORT structure

6/19/2020 • 2 minutes to read • [Edit Online](#)

Indicates the level of support for protected resource sessions.

Syntax

```
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

Header	d3d12.h

D3D12_FEATURE_DATA_QUERY_META_COMMAND structure

6/19/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	File
	d3d12.h

D3D12_FEATURE_DATA_ROOT_SIGNATURE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates root signature version support.

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

[Root Signature Version 1.1](#)

D3D12_FEATURE_DATA_SERIALIZATION structure

6/19/2020 • 2 minutes to read • [Edit Online](#)

Indicates the level of support for heap serialization.

Syntax

```
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

Header	d3d12.h

D3D12_FEATURE_DATA_SHADER_CACHE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef struct D3D12_FEATURE_DATA_SHADER_CACHE {
    D3D12_SHADER_CACHE_SUPPORT_FLAGS SupportFlags;
} D3D12_FEATURE_DATA_SHADER_CACHE;
```

Members

`SupportFlags`

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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FEATURE_DATA_SHADER_MODEL structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Contains the supported shader model.

Syntax

```
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](#) enumeration.

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.

Requirements

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_FEATURE](#)

D3D12_FENCE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies fence options.

Syntax

```
typedef enum D3D12_FENCE_FLAGS {  
    D3D12_FENCE_FLAG_NONE,  
    D3D12_FENCE_FLAG_SHARED,  
    D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER,  
    D3D12_FENCE_FLAG_NON_MONITORED  
} ;
```

Constants

D3D12_FENCE_FLAG_NONE	No options are specified.
D3D12_FENCE_FLAG_SHARED	The fence is shared.
D3D12_FENCE_FLAG_SHARED_CROSS_ADAPTER	The fence is shared with another GPU adapter.
D3D12_FENCE_FLAG_NON_MONITORED	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

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

See also

[Core Enumerations](#)

D3D12_FILL_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the fill mode to use when rendering triangles.

Syntax

```
typedef enum D3D12_FILL_MODE {  
    D3D12_FILL_MODE_WIREFRAME,  
    D3D12_FILL_MODE_SOLID  
} ;
```

Constants

D3D12_FILL_MODE_WIREFRAME	Draw lines connecting the vertices. Adjacent vertices are not drawn.
D3D12_FILL_MODE_SOLID	Fill the triangles formed by the vertices. Adjacent vertices are not drawn.

Remarks

Fill mode is specified in a [D3D12_RASTERIZER_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[CD3DX12_RASTERIZER_DESC](#)

[Core Enumerations](#)

D3D12_FILTER enumeration

5/27/2020 • 7 minutes to read • [Edit Online](#)

Specifies filtering options during texture sampling.

Syntax

```
typedef enum D3D12_FILTER {  
    D3D12_FILTER_MIN_MAG_MIP_POINT,  
    D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR,  
    D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT,  
    D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MIN_MAG_MIP_LINEAR,  
    D3D12_FILTER_ANISOTROPIC,  
    D3D12_FILTER_COMPARISON_MIN_MAG_MIP_POINT,  
    D3D12_FILTER_COMPARISON_MIN_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_COMPARISON_MIN_POINT_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_COMPARISON_MIN_POINT_MAG_MIP_LINEAR,  
    D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_MIP_POINT,  
    D3D12_FILTER_COMPARISON_MIN_LINEAR_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_COMPARISON_MIN_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_COMPARISON_MIN_MAG_MIP_LINEAR,  
    D3D12_FILTER_COMPARISON_ANISOTROPIC,  
    D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT,  
    D3D12_FILTER_MINIMUM_MIN_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MINIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MINIMUM_MIN_POINT_MAG_MIP_LINEAR,  
    D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_MIP_POINT,  
    D3D12_FILTER_MINIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MINIMUM_MIN_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MINIMUM_MIN_MAG_MIP_LINEAR,  
    D3D12_FILTER_MINIMUM_ANISOTROPIC,  
    D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_POINT,  
    D3D12_FILTER_MAXIMUM_MIN_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MAXIMUM_MIN_POINT_MAG_MIP_LINEAR,  
    D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_MIP_POINT,  
    D3D12_FILTER_MAXIMUM_MIN_LINEAR_MAG_POINT_MIP_LINEAR,  
    D3D12_FILTER_MAXIMUM_MIN_MAG_LINEAR_MIP_POINT,  
    D3D12_FILTER_MAXIMUM_MIN_MAG_MIP_LINEAR,  
    D3D12_FILTER_MAXIMUM_ANISOTROPIC  
};
```

Constants

D3D12_FILTER_MIN_MAG_MIP_POINT	Use point sampling for minification, magnification, and mip-level sampling.
D3D12_FILTER_MIN_MAG_POINT_MIP_LINEAR	Use point sampling for minification and magnification; use linear interpolation for mip-level sampling.

D3D12_FILTER_MIN_POINT_MAG_LINEAR_MIP_POINT	Use point sampling for minification; use linear interpolation for magnification; use point sampling for mip-level sampling.
D3D12_FILTER_MIN_POINT_MAG_MIP_LINEAR	Use point sampling for minification; use linear interpolation for magnification and mip-level sampling.
D3D12_FILTER_MIN_LINEAR_MAG_MIP_POINT	Use linear interpolation for minification; use point sampling for magnification and mip-level sampling.
D3D12_FILTER_MIN_LINEAR_MAG_POINT_MIP_LINEAR	Use linear interpolation for minification; use point sampling for magnification; use linear interpolation for mip-level sampling.
D3D12_FILTER_MIN_MAG_LINEAR_MIP_POINT	Use linear interpolation for minification and magnification; use point sampling for mip-level sampling.
D3D12_FILTER_MIN_MAG_MIP_LINEAR	Use linear interpolation for minification, magnification, and mip-level sampling.
D3D12_FILTER_ANISOTROPIC	Use anisotropic interpolation for minification, magnification, and mip-level sampling.
D3D12_FILTER_COMPARISON_MIN_MAG_MIP_POINT	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	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	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	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	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	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	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	Use linear interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.
D3D12_FILTER_COMPARISON_ANISOTROPIC	Use anisotropic interpolation for minification, magnification, and mip-level sampling. Compare the result to the comparison value.

D3D12_FILTER_MINIMUM_MIN_MAG_MIP_POINT	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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:

```

#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 ) ) )

```

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

Header	d3d12.h

See also

[Core Enumerations](#)

[Descriptors](#)

D3D12_FILTER_REDUCTION_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of filter reduction.

Syntax

```
typedef enum D3D12_FILTER_REDUCTION_TYPE {  
    D3D12_FILTER_REDUCTION_TYPE_STANDARD,  
    D3D12_FILTER_REDUCTION_TYPE_COMPARISON,  
    D3D12_FILTER_REDUCTION_TYPE_MINIMUM,  
    D3D12_FILTER_REDUCTION_TYPE_MAXIMUM  
} ;
```

Constants

D3D12_FILTER_REDUCTION_TYPE_STANDARD	The filter type is standard.
D3D12_FILTER_REDUCTION_TYPE_COMPARISON	The filter type is comparison.
D3D12_FILTER_REDUCTION_TYPE_MINIMUM	The filter type is minimum.
D3D12_FILTER_REDUCTION_TYPE_MAXIMUM	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

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

See also

[Core Enumerations](#)

D3D12_FILTER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of magnification or minification sampler filters.

Syntax

```
typedef enum D3D12_FILTER_TYPE {  
    D3D12_FILTER_TYPE_POINT,  
    D3D12_FILTER_TYPE_LINEAR  
} ;
```

Constants

D3D12_FILTER_TYPE_POINT	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.
D3D12_FILTER_TYPE_LINEAR	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_FORMAT_SUPPORT1 enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies resources that are supported for a provided format.

Syntax

```
typedef enum D3D12_FORMAT_SUPPORT1 {
    D3D12_FORMAT_SUPPORT1_NONE,
    D3D12_FORMAT_SUPPORT1_BUFFER,
    D3D12_FORMAT_SUPPORT1_IA_VERTEX_BUFFER,
    D3D12_FORMAT_SUPPORT1_IA_INDEX_BUFFER,
    D3D12_FORMAT_SUPPORT1_SO_BUFFER,
    D3D12_FORMAT_SUPPORT1_TEXTURE1D,
    D3D12_FORMAT_SUPPORT1_TEXTURE2D,
    D3D12_FORMAT_SUPPORT1_TEXTURE3D,
    D3D12_FORMAT_SUPPORT1_TEXTURECUBE,
    D3D12_FORMAT_SUPPORT1_SHADER_LOAD,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_COMPARISON,
    D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_MONO_TEXT,
    D3D12_FORMAT_SUPPORT1_MIP,
    D3D12_FORMAT_SUPPORT1_RENDER_TARGET,
    D3D12_FORMAT_SUPPORT1_BLENDABLE,
    D3D12_FORMAT_SUPPORT1_DEPTH_STENCIL,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RESOLVE,
    D3D12_FORMAT_SUPPORT1_DISPLAY,
    D3D12_FORMAT_SUPPORT1_CAST_WITHIN_BIT_LAYOUT,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RENDERTARGET,
    D3D12_FORMAT_SUPPORT1_MULTISAMPLE_LOAD,
    D3D12_FORMAT_SUPPORT1_SHADER_GATHER,
    D3D12_FORMAT_SUPPORT1_BACK_BUFFER_CAST,
    D3D12_FORMAT_SUPPORT1_TYPED_UNORDERED_ACCESS_VIEW,
    D3D12_FORMAT_SUPPORT1_SHADER_GATHER_COMPARISON,
    D3D12_FORMAT_SUPPORT1_DECODER_OUTPUT,
    D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_OUTPUT,
    D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_INPUT,
    D3D12_FORMAT_SUPPORT1_VIDEO_ENCODER
} ;
```

Constants

D3D12_FORMAT_SUPPORT1_NONE	No resources are supported.
D3D12_FORMAT_SUPPORT1_BUFFER	Buffer resources supported.
D3D12_FORMAT_SUPPORT1_IA_VERTEX_BUFFER	Vertex buffers supported.
D3D12_FORMAT_SUPPORT1_IA_INDEX_BUFFER	Index buffers supported.
D3D12_FORMAT_SUPPORT1_SO_BUFFER	Streaming output buffers supported.
D3D12_FORMAT_SUPPORT1_TEXTURE1D	1D texture resources supported.

D3D12_FORMAT_SUPPORT1_TEXTURE2D	2D texture resources supported.
D3D12_FORMAT_SUPPORT1_TEXTURE3D	3D texture resources supported.
D3D12_FORMAT_SUPPORT1_TEXTURECUBE	Cube texture resources supported.
D3D12_FORMAT_SUPPORT1_SHADER_LOAD	The HLSL Load function for texture objects is supported.
D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE	The HLSL Sample function for texture objects is supported. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> 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. </div>
D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_COMPARISON	The HLSL SampleCmp and SampleCmpLevelZero functions for texture objects are supported. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> 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. </div>
D3D12_FORMAT_SUPPORT1_SHADER_SAMPLE_MONO_TEXT	Reserved.
D3D12_FORMAT_SUPPORT1_MIP	Mipmaps are supported.
D3D12_FORMAT_SUPPORT1_RENDER_TARGET	Render targets are supported.
D3D12_FORMAT_SUPPORT1_BLENDABLE	Blend operations supported.
D3D12_FORMAT_SUPPORT1_DEPTH_STENCIL	Depth stencils supported.
D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RESOLVE	Multisample antialiasing (MSAA) resolve operations are supported. For more info, see ID3D12GraphicsCommandList::ResolveSubresource .
D3D12_FORMAT_SUPPORT1_DISPLAY	Format can be displayed on screen.
D3D12_FORMAT_SUPPORT1_CAST_WITHIN_BIT_LAYOUT	Format can't be cast to another format.
D3D12_FORMAT_SUPPORT1_MULTISAMPLE_RENDERTARGET	Format can be used as a multi-sampled render target.
D3D12_FORMAT_SUPPORT1_MULTISAMPLE_LOAD	Format can be used as a multi-sampled texture and read into a shader with the HLSL Load function.

D3D12_FORMAT_SUPPORT1_SHADER_GATHER	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	Format supports casting when the resource is a back buffer.
D3D12_FORMAT_SUPPORT1_TYPED_UNORDERED_ACCESS_VIEW	Format can be used for an unordered access view.
D3D12_FORMAT_SUPPORT1_SHADER_GATHER_COMPARISON	Format can be used with the HLSL gather with comparison function.
D3D12_FORMAT_SUPPORT1_DECODER_OUTPUT	Format can be used with the decoder output.
D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_OUTPUT	Format can be used with the video processor output.
D3D12_FORMAT_SUPPORT1_VIDEO_PROCESSOR_INPUT	Format can be used with the video processor input.
D3D12_FORMAT_SUPPORT1_VIDEO_ENCODER	Format can be used with the video encoder.

Remarks

This enum is used by the [D3D12_FEATURE_DATA_FORMAT_SUPPORT](#) structure.

Requirements

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

See also

[Core Enumerations](#)

[D3D12_HEAP_FLAGS](#)

D3D12_FORMAT_SUPPORT2 enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies which unordered resource options are supported for a provided format.

Syntax

```
typedef enum D3D12_FORMAT_SUPPORT2 {
    D3D12_FORMAT_SUPPORT2_NONE,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_ADD,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX,
    D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX,
    D3D12_FORMAT_SUPPORT2_UAV_TYPED_LOAD,
    D3D12_FORMAT_SUPPORT2_UAV_TYPED_STORE,
    D3D12_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP,
    D3D12_FORMAT_SUPPORT2_TILED,
    D3D12_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY,
    D3D12_FORMAT_SUPPORT2_SAMPLER_FEEDBACK
} ;
```

Constants

D3D12_FORMAT_SUPPORT2_NONE	No unordered resource options are supported.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_ADD	Format supports atomic add.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_BITWISE_OPS	Format supports atomic bitwise operations.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_COMPARE_STORE_OR_COMPARE_EXCHANGE	Format supports atomic compare with store or exchange.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_EXCHANGE	Format supports atomic exchange.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_SIGNED_MIN_OR_MAX	Format supports atomic min and max.
D3D12_FORMAT_SUPPORT2_UAV_ATOMIC_UNSIGNED_MIN_OR_MAX	Format supports atomic unsigned min and max.
D3D12_FORMAT_SUPPORT2_UAV_TYPED_LOAD	Format supports a typed load.
D3D12_FORMAT_SUPPORT2_UAV_TYPED_STORE	Format supports a typed store.
D3D12_FORMAT_SUPPORT2_OUTPUT_MERGER_LOGIC_OP	Format supports logic operations in blend state.
D3D12_FORMAT_SUPPORT2_TILED	Format supports tiled resources. Refer to Volume Tiled Resources .

D3D12_FORMAT_SUPPORT2_MULTIPLANE_OVERLAY	Format supports multi-plane overlays.
--	---------------------------------------

Remarks

This enum is used by the [D3D12_FEATURE_DATA_FORMAT_SUPPORT](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_GLOBAL_ROOT_SIGNATURE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

D3D12_GPU_DESCRIPTOR_HANDLE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a GPU descriptor handle.

Syntax

```
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](#)

Requirements

Header	d3d12.h

See also

[CD3DX12_GPU_DESCRIPTOR_HANDLE](#)

[Core Structures](#)

D3D12_GPU_VIRTUAL_ADDRESS_AND_STRIDE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a GPU virtual address and indexing stride.

Syntax

```
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

Header

d3d12.h

D3D12_GPU_VIRTUAL_ADDRESS_RANGE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a GPU virtual address range.

Syntax

```
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

Header	d3d12.h

D3D12_GPU_VIRTUAL_ADDRESS_RANGE_AND_STRIDE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a GPU virtual address range and stride.

Syntax

```
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

Header	d3d12.h

D3D12_GRAPHICS_PIPELINE_STATE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a graphics pipeline state object.

Syntax

```
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`

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.

`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

Header	
	d3d12.h

See also

[Conservative Rasterization](#)

[Core Structures](#)

[Rasterizer Ordered Views](#)

D3D12_GRAPHICS_STATES enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines flags that specify states related to a graphics command list. Values can be bitwise OR'd together.

Syntax

```
typedef enum D3D12_GRAPHICS_STATES {
    D3D12_GRAPHICS_STATE_NONE,
    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

D3D12_GRAPHICS_STATE_NONE	Specifies no state.
D3D12_GRAPHICS_STATE_IA_VERTEX_BUFFERS	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

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

D3D12_HEAP_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a heap.

Syntax

```
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:

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

Header	d3d12.h

See also

[CD3DX12_HEAP_DESC](#)

[Core Structures](#)

[Descriptor Heaps](#)

D3D12_HEAP_FLAGS enumeration

5/13/2020 • 3 minutes to read • [Edit Online](#)

Specifies heap options, such as whether the heap can contain textures, and whether resources are shared across adapters.

Syntax

```
typedef enum D3D12_HEAP_FLAGS {  
    D3D12_HEAP_FLAG_NONE,  
    D3D12_HEAP_FLAG_SHARED,  
    D3D12_HEAP_FLAG_DENY_BUFFERS,  
    D3D12_HEAP_FLAG_ALLOW_DISPLAY,  
    D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER,  
    D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES,  
    D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES,  
    D3D12_HEAP_FLAG_HARDWARE_PROTECTED,  
    D3D12_HEAP_FLAG_ALLOW_WRITE_WATCH,  
    D3D12_HEAP_FLAG_ALLOW_SHADER_ATOMICS,  
    D3D12_HEAP_FLAG_CREATE_NOT_RESIDENT,  
    D3D12_HEAP_FLAG_CREATE_NOT_ZEROED,  
    D3D12_HEAP_FLAG_ALLOW_ALL_BUFFERS_AND_TEXTURES,  
    D3D12_HEAP_FLAG_ALLOW_ONLY_BUFFERS,  
    D3D12_HEAP_FLAG_ALLOW_ONLY_NON_RT_DS_TEXTURES,  
    D3D12_HEAP_FLAG_ALLOW_ONLY_RT_DS_TEXTURES  
} ;
```

Constants

Constant	Description
D3D12_HEAP_FLAG_NONE	No options are specified.
D3D12_HEAP_FLAG_SHARED	The heap is shared. Refer to Shared Heaps .
D3D12_HEAP_FLAG_DENY_BUFFERS	The heap isn't allowed to contain buffers.
D3D12_HEAP_FLAG_ALLOW_DISPLAY	The heap is allowed to contain swap-chain surfaces.
D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER	The heap is allowed to share resources across adapters. Refer to Shared Heaps .
D3D12_HEAP_FLAG_DENY_RT_DS_TEXTURES	The heap is not allowed to store Render Target (RT) and/or Depth-Stencil (DS) textures.
D3D12_HEAP_FLAG_DENY_NON_RT_DS_TEXTURES	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	Unsupported. Do not use.
D3D12_HEAP_FLAG_ALLOW_WRITE_WATCH	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	<p>Ensures that atomic operations will be atomic on this heap's memory, according to components able to see the memory.</p> <p>Creating a heap with this flag will fail under either of these conditions.</p> <ul style="list-style-type: none"> - The heap type is D3D12_HEAP_TYPE_DEFAULT, and the heap can be visible on multiple nodes, but the device does <i>not</i> support D3D12_CROSS_NODE_SHARING_TIER_3. - The heap is CPU-visible, but the heap type is <i>not</i> D3D12_HEAP_TYPE_CUSTOM. <p>Note that heaps with this flag might be a limited resource on some systems.</p>
D3D12_HEAP_FLAG_ALLOW_ALL_BUFFERS_AND_TEXTURES	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	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	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	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.

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

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

See also

[CD3DX12_HEAP_DESC](#)

[Core Enumerations](#)

[Descriptor Heaps](#)

D3D12_HEAP_PROPERTIES structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes heap properties.

Syntax

```
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 bits set as `CreationNodeMask` has. See [Multi-adapter systems](#).

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

Header	
	d3d12.h

See also

[CD3DX12_HEAP_PROPERTIES](#)

[Core Structures](#)

[Descriptor Heaps](#)

D3D12_HEAP_TYPE enumeration

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_HEAP_TYPE {  
    D3D12_HEAP_TYPE_DEFAULT,  
    D3D12_HEAP_TYPE_UPLOAD,  
    D3D12_HEAP_TYPE_READBACK,  
    D3D12_HEAP_TYPE_CUSTOM  
} ;
```

Constants

D3D12_HEAP_TYPE_DEFAULT

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	<p>Specifies a heap used for uploading.</p> <p>This heap type has CPU access optimized for uploading to the GPU, but does not experience the maximum amount of bandwidth for the GPU.</p> <p>This heap type is best for CPU-write-once, GPU-read-once data; but GPU-read-once is stricter than necessary.</p> <p>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.</p> <p>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.</p> <p>Resources in this heap must be created with D3D12_RESOURCE_STATE_GENERIC_READ and cannot be changed away from this.</p> <p>The CPU address for such heaps is commonly not efficient for CPU reads.</p> <p>The following are typical usages for _UPLOAD heaps:</p> <ul style="list-style-type: none"> • 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. <p>The following are likely not good usages for _UPLOAD heaps:</p> <ul style="list-style-type: none"> • 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	<p>Specifies a heap used for reading back.</p> <p>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.</p> <p>This heap type is best for GPU-write-once, CPU-readable data.</p> <p>The CPU cache behavior is write-back, which is conducive for multiple sub-cache-line CPU reads.</p> <p>Resources in this heap must be created with D3D12_RESOURCE_STATE_COPY_DEST, and cannot be changed away from this.</p>

D3D12_HEAP_TYPE_CUSTOM	<p>Specifies a custom heap.</p> <p>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.</p> <p>To do so, the application is expected to understand the adapter architecture to make the right choice.</p> <p>For more details, see D3D12_FEATURE_ARCHITECTURE, D3D12_FEATURE_DATA_ARCHITECTURE, and GetCustomHeapProperties.</p>
------------------------	--

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

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

See also

[Core Enumerations](#)

[Descriptor Heaps](#)

D3D12_HIT_GROUP_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

D3D12_HIT_GROUP_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of a raytracing hit group state subobject. Use a value from this enumeration with the [D3D12_HIT_GROUP_DESC](#) structure.

Syntax

```
typedef enum D3D12_HIT_GROUP_TYPE {
    D3D12_HIT_GROUP_TYPE_TRIANGLES,
    D3D12_HIT_GROUP_TYPE PROCEDURAL_PRIMITIVE
};
```

Constants

D3D12_HIT_GROUP_TYPE_TRIANGLES	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	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

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

D3D12_INDEX_BUFFER_STRIP_CUT_VALUE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_INDEX_BUFFER_STRIP_CUT_VALUE {
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_DISABLED,
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFF,
    D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFFFFFF
} ;
```

Constants

D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_DISABLED	Indicates that there is no cut value.
D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFF	Indicates that 0xFFFF should be used as the cut value.
D3D12_INDEX_BUFFER_STRIP_CUT_VALUE_0xFFFFFFFF	Indicates that 0xFFFFFFFF should be used as the cut value.

Remarks

This enum is used by the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_INDEX_BUFFER_VIEW structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the index buffer to view.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_INDIRECT_ARGUMENT_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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;
    };
} 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.

Remarks

Use this structure with the [D3D12_COMMAND_SIGNATURE_DESC](#) structure.

Requirements

Header	d3d12.h

See also

[Core Structures](#)

[Example Root Signatures](#)

D3D12_INDIRECT_ARGUMENT_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of the indirect parameter.

Syntax

```
typedef enum D3D12_INDIRECT_ARGUMENT_TYPE {
    D3D12_INDIRECT_ARGUMENT_TYPE_DRAW,
    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
} ;
```

Constants

D3D12_INDIRECT_ARGUMENT_TYPE_DRAW	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

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

See also

[Core Enumerations](#)

D3D12_INPUT_CLASSIFICATION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the type of data contained in an input slot.

Syntax

```
typedef enum D3D12_INPUT_CLASSIFICATION {
    D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA,
    D3D12_INPUT_CLASSIFICATION_PER_INSTANCE_DATA
} ;
```

Constants

D3D12_INPUT_CLASSIFICATION_PER_VERTEX_DATA	Input data is per-vertex data.
D3D12_INPUT_CLASSIFICATION_PER_INSTANCE_DATA	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

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

See also

[Core Enumerations](#)

D3D12_INPUT_ELEMENT_DESC structure

4/25/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_INPUT_LAYOUT_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

D3D12_LOCAL_ROOT_SIGNATURE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header

d3d12.h

D3D12_LOGIC_OP enumeration

6/25/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify logical operations to configure for a render target.

Syntax

```
typedef enum D3D12_LOGIC_OP {
    D3D12_LOGIC_OP_CLEAR,
    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

D3D12_LOGIC_OP_CLEAR	Clears the render target (<code>s</code>).
D3D12_LOGIC_OP_SET	Sets the render target (<code>d</code>).
D3D12_LOGIC_OP_COPY	Copies the render target (<small><code>s</code> source from Pixel Shader output</small>).
D3D12_LOGIC_OP_COPY_INVERTED	Performs an inverted-copy of the render target (<code>~s</code>).
D3D12_LOGIC_OP_NOOP	No operation is performed on the render target (<code>d</code> destination in the Render Target View).
D3D12_LOGIC_OP_INVERT	Inverts the render target (<code>~d</code>).
D3D12_LOGIC_OP_AND	Performs a logical AND operation on the render target (<code>s & d</code>).
D3D12_LOGIC_OP_NAND	Performs a logical NAND operation on the render target (<code>~(s & d)</code>).

D3D12_LOGIC_OP_OR	Performs a logical OR operation on the render target ($s \vee d$).	d).
D3D12_LOGIC_OP_NOR	Performs a logical NOR operation on the render target ($\neg(s \vee d)$).	d)).
D3D12_LOGIC_OP_XOR	Performs a logical XOR operation on the render target ($s \wedge \neg d$).	
D3D12_LOGIC_OP_EQUIV	Performs a logical equal operation on the render target ($\neg(s \wedge \neg d)$).	
D3D12_LOGIC_OP_AND_REVERSE	Performs a logical AND and reverse operation on the render target ($s \wedge \neg d$).	
D3D12_LOGIC_OP_AND_INVERTED	Performs a logical AND and invert operation on the render target ($\neg s \wedge d$).	
D3D12_LOGIC_OP_OR_REVERSE	Performs a logical OR and reverse operation on the render target ($\neg s \vee d$).	$\neg d$.
D3D12_LOGIC_OP_OR_INVERTED	Performs a logical OR and invert operation on the render target ($\neg s \vee \neg d$).	d).

Remarks

This enum is used by the [D3D12_RENDER_TARGET_BLEND_DESC](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_MEASUREMENTS_ACTION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_MEASUREMENTS_ACTION {
    D3D12_MEASUREMENTS_ACTION_KEEP_ALL,
    D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS,
    D3D12_MEASUREMENTS_ACTION_COMMIT_RESULTS_HIGH_PRIORITY,
    D3D12_MEASUREMENTS_ACTION_DISCARD_PREVIOUS
} ;
```

Constants

D3D12_MEASUREMENTS_ACTION_KEEP_ALL	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

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

See also

[Core enumerations](#)

D3D12_MEMCPY_DEST structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the destination of a memory copy operation.

Syntax

```
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

Header	File
d3d12.h	

See also

[Core Structures](#)

D3D12_MEMORY_POOL enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the memory pool for the heap.

Syntax

```
typedef enum D3D12_MEMORY_POOL {  
    D3D12_MEMORY_POOL_UNKNOWN,  
    D3D12_MEMORY_POOL_L0,  
    D3D12_MEMORY_POOL_L1  
} ;
```

Constants

D3D12_MEMORY_POOL_UNKNOWN	The memory pool is unknown.
D3D12_MEMORY_POOL_L0	<p>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.</p>
D3D12_MEMORY_POOL_L1	<p>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.</p>

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

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

See also

[Core Enumerations](#)

[Descriptor Heaps](#)

D3D12_META_COMMAND_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a meta command.

Syntax

```
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

Header

d3d12.h

D3D12_META_COMMAND_PARAMETER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a parameter to a meta command.

Syntax

```
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

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

D3D12_META_COMMAND_PARAMETER_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_META_COMMAND_PARAMETER_FLAGS {  
    D3D12_META_COMMAND_PARAMETER_FLAG_INPUT,  
    D3D12_META_COMMAND_PARAMETER_FLAG_OUTPUT  
} ;
```

Constants

D3D12_META_COMMAND_PARAMETER_FLAG_INPUT	Specifies that the parameter is an input resource.
D3D12_META_COMMAND_PARAMETER_FLAG_OUTPUT	Specifies that the parameter is an output resource.

Requirements

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

D3D12_META_COMMAND_PARAMETER_STAGE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_META_COMMAND_PARAMETER_STAGE {
    D3D12_META_COMMAND_PARAMETER_STAGE_CREATION,
    D3D12_META_COMMAND_PARAMETER_STAGE_INITIALIZATION,
    D3D12_META_COMMAND_PARAMETER_STAGE_EXECUTION
} ;
```

Constants

D3D12_META_COMMAND_PARAMETER_STAGE_CREATION	Specifies that the parameter is used at the meta command creation stage.
D3D12_META_COMMAND_PARAMETER_STAGE_INITIALIZATION	Specifies that the parameter is used at the meta command initialization stage.
D3D12_META_COMMAND_PARAMETER_STAGE_EXECUTION	Specifies that the parameter is used at the meta command execution stage.

Requirements

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

D3D12_META_COMMAND_PARAMETER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_META_COMMAND_PARAMETER_TYPE {  
    D3D12_META_COMMAND_PARAMETER_TYPE_FLOAT,  
    D3D12_META_COMMAND_PARAMETER_TYPE_UINT64,  
    D3D12_META_COMMAND_PARAMETER_TYPE_GPU_VIRTUAL_ADDRESS,  
    D3D12_META_COMMAND_PARAMETER_TYPE_CPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UAV,  
    D3D12_META_COMMAND_PARAMETER_TYPE_GPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UAV  
} ;
```

Constants

D3D12_META_COMMAND_PARAMETER_TYPE_FLOAT	Specifies that the parameter is of type FLOAT .
D3D12_META_COMMAND_PARAMETER_TYPE_UINT64	Specifies that the parameter is of type UINT64 .
D3D12_META_COMMAND_PARAMETER_TYPE_GPU_VIRTUAL_ADDRESS	Specifies that the parameter is a GPU virtual address.
D3D12_META_COMMAND_PARAMETER_TYPE_CPU_DESCRIPTOR_HANDLE_HEAP_TYPE_CBV_SRV_UAV	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	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

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

D3D12_MULTIPLE_FENCE_WAIT_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies multiple wait flags for multiple fences.

Syntax

```
typedef enum D3D12_MULTIPLE_FENCE_WAIT_FLAGS {  
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_NONE,  
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_ANY,  
    D3D12_MULTIPLE_FENCE_WAIT_FLAG_ALL  
} ;
```

Constants

D3D12_MULTIPLE_FENCE_WAIT_FLAG_NONE	No flags are being passed. This means to use the default behavior, which is to wait for all fences before signaling the event.
D3D12_MULTIPLE_FENCE_WAIT_FLAG_ANY	Modifies behavior to indicate that the event should be signaled after any one of the fence values has been reached by its corresponding fence.
D3D12_MULTIPLE_FENCE_WAIT_FLAG_ALL	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

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

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies options for determining quality levels.

Syntax

```
typedef enum D3D12_MULTISAMPLE_QUALITY_LEVEL_FLAGS {  
    D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_NONE,  
    D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_TILED_RESOURCE  
} ;
```

Constants

D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_NONE	No options are supported.
D3D12_MULTISAMPLE_QUALITY_LEVELS_FLAG_TILED_RESOURCE	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_NODE_MASK structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

Requirements

Header	
d3d12.h	

D3D12_PACKED_MIP_INFO structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the tile structure of a tiled resource with mipmaps.

Syntax

```
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

Header	d3d12.h

See also

[CD3DX12_PACKED_MIP_INFO](#)

[Core Structures](#)

D3D12_PIPELINE_STATE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Flags to control pipeline state.

Syntax

```
typedef enum D3D12_PIPELINE_STATE_FLAGS {  
    D3D12_PIPELINE_STATE_FLAG_NONE,  
    D3D12_PIPELINE_STATE_FLAG_TOOL_DEBUG  
} ;
```

Constants

D3D12_PIPELINE_STATE_FLAG_NONE	Indicates no flags.
D3D12_PIPELINE_STATE_FLAG_TOOL_DEBUG	Indicates that the pipeline state should be compiled with additional information to assist debugging. This can only be set on WARP devices.

Remarks

This enum is used by the **Flags** member of the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) and [D3D12_COMPUTE_PIPELINE_STATE_DESC](#) structures.

Requirements

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_PIPELINE_STATE_STREAM_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a pipeline state stream.

Syntax

```
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("Dependent on size of subobjects"))*

Specifies the address of a data structure that describes as a bytestream an arbitrary pipeline state subobject.

Remarks

Use this structure with the ID3D12Device1::CreatePipelineState method to create pipeline state objects.

Requirements

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

See also

[Core Structures](#)

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_PIPELINE_STATE_SUBOBJECT_TYPE {
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_ROOT_SIGNATURE,
    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,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MS,
    D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MAX_VALID
} ;
```

Constants

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_ROOT_SIGNATURE	Indicates a root signature subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VS	Indicates a vertex shader subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PS	Indicates a pixel shader subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DS	Indicates a domain shader subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_HS	Indicates a hull shader subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_GS	Indicates a geometry shader subobject type.

D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CS	Indicates a compute shader subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_STREAM_OUTPUT	Indicates a stream-output subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_BLEND	Indicates a blend subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_MASK	Indicates a sample mask subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RASTERIZER	Indicates indicates a rasterizer subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_DEPTH_STENCIL	Indicates a depth stencil subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_INPUT_LAYOUT	Indicates an input layout subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_IB_STRIP_CUT_VALUE	Indicates an index buffer strip cut value subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_PRIMITIVE_TOPOLOGY	Indicates a primitive topology subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_RENDER_TARGET_FORMATS	Indicates a render target formats subobject type. This subobject type corresponds to the 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.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_SAMPLE_DESC	Indicates a sample description subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_NODE_MASK	Indicates a node mask subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_CACHED_PSO	Indicates a cached pipeline state object subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_FLAGS	Indicates a flags subobject type.
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.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_VIEW_INSTANCING	Indicates a view instancing subobject type.
D3D12_PIPELINE_STATE_SUBOBJECT_TYPE_MAX_VALID	A sentinel value that marks the exclusive upper-bound of valid values this enumeration represents.

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

Header	
	d3d12.h

See also

[Core Enumerations](#)

D3D12_PLACED_SUBRESOURCE_FOOTPRINT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_PREDICATION_OP enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the predication operation to apply.

Syntax

```
typedef enum D3D12_PREDICATION_OP {
    D3D12_PREDICATION_OP_EQUAL_ZERO,
    D3D12_PREDICATION_OP_NOT_EQUAL_ZERO
} ;
```

Constants

D3D12_PREDICATION_OP_EQUAL_ZERO	Enables predication if all 64-bits are zero.
D3D12_PREDICATION_OP_NOT_EQUAL_ZERO	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

Header	d3d12.h

See also

[Core Enumerations](#)

[Predication](#)

D3D12_PRIMITIVE_TOPOLOGY_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_PRIMITIVE_TOPOLOGY_TYPE {
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_UNDEFINED,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_POINT,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_LINE,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE,
    D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH
} ;
```

Constants

D3D12_PRIMITIVE_TOPOLOGY_TYPE_UNDEFINED	The shader has not been initialized with an input primitive type.
D3D12_PRIMITIVE_TOPOLOGY_TYPE_POINT	Interpret the input primitive as a point.
D3D12_PRIMITIVE_TOPOLOGY_TYPE_LINE	Interpret the input primitive as a line.
D3D12_PRIMITIVE_TOPOLOGY_TYPE_TRIANGLE	Interpret the input primitive as a triangle.
D3D12_PRIMITIVE_TOPOLOGY_TYPE_PATCH	Interpret the input primitive as a control point patch.

Remarks

This enum is used by the [D3D12_GRAPHICS_PIPELINE_STATE_DESC](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER enumeration

4/22/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER {  
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_NOT_SUPPORTED,  
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_1,  
    D3D12_PROGRAMMABLE_SAMPLE_POSITIONS_TIER_2  
} ;
```

Constants

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 <i>ResolveSubResource</i> 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> = 1) 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

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

See also

[Core Enumerations](#)

D3D12_PROTECTED_RESOURCE_SESSION_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes flags for a protected resource session, per adapter.

Syntax

```
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.

Flags

Type: [D3D12_PROTECTED_RESOURCE_SESSION_FLAGS](#)

Flags.

Requirements

Header	d3d12.h

D3D12_PROTECTED_RESOURCE_SESSION_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify protected resource session flags. These flags can be bitwise OR'd together to specify multiple flags at once.

Syntax

```
typedef enum D3D12_PROTECTED_RESOURCE_SESSION_FLAGS {  
    D3D12_PROTECTED_RESOURCE_SESSION_FLAG_NONE  
} ;
```

Constants

D3D12_PROTECTED_RESOURCE_SESSION_FLAG_NONE	Specifies no flag.
--	--------------------

Requirements

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

D3D12_QUERY_DATA_PIPELINE_STATISTICS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Query information about graphics-pipeline activity in between calls to [BeginQuery](#) and [EndQuery](#).

Syntax

```
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.

VSIgnovations

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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_QUERY_DATA_SO_STATISTICS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes query data for stream output.

Syntax

```
typedef struct D3D12_QUERY_DATA_SO_STATISTICS {  
    UINT64 NumPrimitivesWritten;  
    UINT64 PrimitivesStorageNeeded;  
} D3D12_QUERY_DATA_SO_STATISTICS;
```

Members

`NumPrimitivesWritten`

Specifies the number of primitives written.

`PrimitivesStorageNeeded`

Specifies the total amount of storage needed by the primitives.

Remarks

Use this structure with [D3D12_QUERY_HEAP_TYPE](#) and [CreateQueryHeap](#).

Requirements

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

D3D12_QUERY_HEAP_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_QUERY_HEAP_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of query heap to create.

Syntax

```
typedef enum D3D12_QUERY_HEAP_TYPE {
    D3D12_QUERY_HEAP_TYPE_OCCLUSION,
    D3D12_QUERY_HEAP_TYPE_TIMESTAMP,
    D3D12_QUERY_HEAP_TYPE_PIPELINE_STATISTICS,
    D3D12_QUERY_HEAP_TYPE_SO_STATISTICS,
    D3D12_QUERY_HEAP_TYPE_VIDEO_DECODE_STATISTICS,
    D3D12_QUERY_HEAP_TYPE_COPY_QUEUE_TIMESTAMP
} ;
```

Constants

D3D12_QUERY_HEAP_TYPE_OCCLUSION	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	Indicates that the heap is for high-performance timing data.
D3D12_QUERY_HEAP_TYPE_PIPELINE_STATISTICS	Indicates the heap is to contain pipeline data. Refer to D3D12_QUERY_DATA_PIPELINE_STATISTICS .
D3D12_QUERY_HEAP_TYPE_SO_STATISTICS	Indicates the heap is to contain stream output data. Refer to D3D12_QUERY_DATA_SO_STATISTICS .
D3D12_QUERY_HEAP_TYPE_VIDEO_DECODE_STATISTICS	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	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

Header	
<code>d3d12.h</code>	

See also

[Core enumerations](#)

D3D12_QUERY_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of query.

Syntax

```
typedef enum D3D12_QUERY_TYPE {
    D3D12_QUERY_TYPE_OCCLUSION,
    D3D12_QUERY_TYPE_BINARY_OCCLUSION,
    D3D12_QUERY_TYPE_TIMESTAMP,
    D3D12_QUERY_TYPE_PIPELINE_STATISTICS,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM0,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM1,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM2,
    D3D12_QUERY_TYPE_SO_STATISTICS_STREAM3,
    D3D12_QUERY_TYPE_VIDEO_DECODE_STATISTICS
} ;
```

Constants

D3D12_QUERY_TYPE_OCCLUSION	Indicates the query is for depth/stencil occlusion counts.
D3D12_QUERY_TYPE_BINARY_OCCLUSION	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	Indicates the query is for high definition GPU and CPU timestamps.
D3D12_QUERY_TYPE_PIPELINE_STATISTICS	Indicates the query type is for graphics pipeline statistics, refer to D3D12_QUERY_DATA_PIPELINE_STATISTICS .
D3D12_QUERY_TYPE_SO_STATISTICS_STREAM0	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	Stream 1 output statistics.
D3D12_QUERY_TYPE_SO_STATISTICS_STREAM2	Stream 2 output statistics.

D3D12_QUERY_TYPE_SO_STATISTICS_STREAM3	Stream 3 output statistics.
D3D12_QUERY_TYPE_VIDEO_DECODE_STATISTICS	<p>Video decode statistics. Refer to D3D12_QUERY_DATA_VIDEO_DECODE_STATISTICS.</p> <p>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.</p> <p>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.</p> <p>Decode status structures are defined by the codec specification.</p>

Remarks

This enum is used by [BeginQuery](#), [EndQuery](#) and [ResolveQueryData](#).

Requirements

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

See also

[Core Enumerations](#)

D3D12_RANGE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a memory range.

Syntax

```
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

Header	d3d12.h

See also

[CD3DX12_RANGE](#)

[Core Structures](#)

D3D12_RANGE_UINT64 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_RASTERIZER_DESC structure

5/27/2020 • 3 minutes to read • [Edit Online](#)

Describes rasterizer state.

Syntax

```
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).

```
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, 2, 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.

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:

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

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

See also

[CD3DX12_RASTERIZER_DESC](#)

[Conservative Rasterization](#)

[Core Structures](#)

[Rasterizer Ordered Views](#)

D3D12_RAY_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Flags passed to the [TraceRay](#) function to override transparency, culling, and early-out behavior.

Syntax

```
typedef enum D3D12_RAY_FLAGS {  
    D3D12_RAY_FLAG_NONE,  
    D3D12_RAY_FLAG_FORCE_OPAQUE,  
    D3D12_RAY_FLAG_FORCE_NON_OPAQUE,  
    D3D12_RAY_FLAG_ACCEPT_FIRST_HIT_AND_END_SEARCH,  
    D3D12_RAY_FLAG_SKIP_CLOSEST_HIT_SHADER,  
    D3D12_RAY_FLAG_CULL_BACK_FACING_TRIANGLES,  
    D3D12_RAY_FLAG_CULL_FRONT_FACING_TRIANGLES,  
    D3D12_RAY_FLAG_CULL_OPAQUE,  
    D3D12_RAY_FLAG_CULL_NON_OPAQUE,  
    D3D12_RAY_FLAG_SKIP_TRIANGLES,  
    D3D12_RAY_FLAG_SKIP_PROCEDURAL_PRIMITIVES  
} ;
```

Constants

D3D12_RAY_FLAG_NONE	No options selected.
D3D12_RAY_FLAG_FORCE_OPAQUE	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	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	<p>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.</p> <p>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.</p> <p>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.</p> <p>A typical use for this flag is for shadows, where only a single hit needs to be found.</p>
D3D12_RAY_FLAG_SKIP_CLOSEST_HIT_SHADER	<p>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.</p>
D3D12_RAY_FLAG_CULL_BACK_FACING_TRIANGLES	<p>Enables culling of back facing triangles. See D3D12_RAYTRACING_INSTANCE_FLAGS for selecting which triangles are back facing, per-instance.</p> <p>On instances that specify D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE, this flag has no effect.</p> <p>On geometry types other than D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES, this flag has no effect.</p> <p>This flag is mutually exclusive with RAY_FLAG_CULL_FRONT_FACING_TRIANGLES.</p>
D3D12_RAY_FLAG_CULL_FRONT_FACING_TRIANGLES	<p>Enables culling of front facing triangles. See D3D12_RAYTRACING_INSTANCE_FLAGS for selecting which triangles are back facing, per-instance.</p> <p>On instances that specify D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE, this flag has no effect.</p> <p>On geometry types other than D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES, this flag has no effect.</p> <p>This flag is mutually exclusive with RAY_FLAG_CULL_BACK_FACING_TRIANGLES.</p>

D3D12_RAY_FLAG_CULL_OPAQUE	<p>Culls all primitives that are considered opaque based on their geometry and instance flags.</p> <p>This flag is mutually exclusive with RAY_FLAG_FORCE_OPAQUE, RAY_FLAG_FORCE_NON_OPAQUE, and RAY_FLAG_CULL_NON_OPAQUE.</p>
D3D12_RAY_FLAG_CULL_NON_OPAQUE	<p>Culls all primitives that are considered non-opaque based on their geometry and instance flags.</p> <p>This flag is mutually exclusive with RAY_FLAG_FORCE_OPAQUE, RAY_FLAG_FORCE_NON_OPAQUE, and RAY_FLAG_CULL_OPAQUE.</p>

Requirements

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

D3D12_RAYTRACING_AABB structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS enumeration

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAGS {
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_NONE,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_TRACE,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_MINIMIZE_MEMORY,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE
} ;
```

Constants

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_NONE	No options specified for the acceleration structure build.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE	<p>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.</p> <p>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.</p>

<p><code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION</code></p>	<p>Enables the option to compact the acceleration structure by calling CopyRaytracingAccelerationStructure using compact mode, specified with <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT</code>.</p> <p>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.</p> <p>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 <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION</code>, no other acceleration structures can be created from it via updates that specify <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_COMPACTION</code>.</p> <p>Specifying ALLOW_COMPACTION may increase pre-compaction acceleration structure size versus not specifying ALLOW_COMPACTION.</p> <p>If multiple incremental builds are performed before finally compacting, there may be redundant compaction related work performed.</p> <p>The size required for the compacted acceleration structure can be queried before compaction via EmitRaytracingAccelerationStructurePostbuildInfo. See <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CCOMPACTED_SIZE_DESC</code> for more information on properties of compacted acceleration structure size.</p> <div style="border: 1px solid black; padding: 5px;"> <p>Note When <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE</code> 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.</p> </div>
<p><code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_TRACE</code></p>	<p>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.</p> <p>This flag is recommended for static geometry in particular. It is compatible with all other flags except for <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD</code>.</p>
<p><code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD</code></p>	<p>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.</p> <p>This flag is compatible with all other flags except for <code>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PREFER_FAST_BUILD</code>.</p>

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_MINIMIZE_MEMORY	<p>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.</p> <p>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.</p> <p>This flag is compatible with all other flags.</p>
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_PERFORM_UPDATE	<p>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.</p> <p>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</p> <p>The source acceleration structure must have been built with D3D12_RAYTRACING_ACCELERATION_STRUCTURE_BUILD_FLAG_ALLOW_UPDATE.</p> <p>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.</p> <p>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.</p>

Requirements

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

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE enumeration

5/27/2020 • 3 minutes to read • [Edit Online](#)

Specifies the type of copy operation performed when calling [CopyRaytracingAccelerationStructure](#).

Syntax

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE {  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_CLONE,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_VISUALIZATION_DECODE_FOR_TOOLS,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_DESERIALIZE  
} ;
```

Constants

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_CLONE	<p>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.</p> <p>The memory pointed to must be in state D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE.</p>
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_COMPACT	<p>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.</p> <p>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.</p> <p>The source and destination memory for the operation must be in state D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE.</p>

<p>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_VISUALIZATION_DECODE_FOR_TOOLS</p>	<p>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.</p> <p>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.</p> <p>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.</p> <p>The source memory for the operation must be in state D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE.</p> <p>The destination memory for the operation must be in state D3D12_RESOURCE_STATE_UNORDERED_ACCESS.</p> <p>This mode is only permitted when developer mode is enabled in the OS.</p>
<p>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_SERIALIZE</p>	<p>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.</p> <p>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.</p> <p>The source memory for the operation must be in state D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE.</p> <p>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.</p>

<p>D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE_DESERALIZE</p>	<p>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.</p> <p>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.</p> <p>Deserialization can only be performed on the same device and driver version on which the data was serialized. Otherwise, the results are undefined.</p> <p>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.</p> <p>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.</p>
---	---

Requirements

<p>Header</p>	<p>d3d12.h</p>
----------------------	----------------

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the space requirement for acceleration structure after compaction.

Syntax

```
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-compact 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

Header	d3d12.h

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the space currently used by an acceleration structure..

Syntax

```
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

Header	
d3d12.h	

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Description of the post-build information to generate from an acceleration structure. Use this structure in calls to [EmitRaytracingAccelerationStructurePostbuildInfo](#) and [BuildRaytracingAccelerationStructure](#).

Syntax

```
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

The type of post-build information to retrieve.

Requirements

Header	d3d12.h

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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 serialized 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

Header	d3d12.h

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef struct D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC {
    UINT64 DecodedSizeInBytes;
} D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION_DESC;
```

Members

`DecodedSizeInBytes`

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

Requirements

Header	
d3d12.h	

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of acceleration structure post-build info that can be retrieved with calls to [EmitRaytracingAccelerationStructurePostbuildInfo](#) and [BuildRaytracingAccelerationStructure](#).

Syntax

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TYPE {  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_TOOLS_VISUALIZATION,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_SERIALIZATION,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_CURRENT_SIZE  
} ;
```

Constants

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_COMPACTED_SIZE	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	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	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	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

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

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents prebuild information about a raytracing acceleration structure. Get an instance of this structure by calling [GetRaytracingAccelerationStructurePrebuildInfo](#).

Syntax

```
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

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

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of a raytracing acceleration structure.

Syntax

```
typedef enum D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE {  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL,  
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL  
} ;
```

Constants

D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_TOP_LEVEL	Top-level acceleration structure.
D3D12_RAYTRACING_ACCELERATION_STRUCTURE_TYPE_BOTTOM_LEVEL	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

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

D3D12_RAYTRACING_GEOMETRY_AABBS_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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 [D3D12_RAYTRACING_AABB_BYTE_ALIGNMENT](#). The address must be aligned to 16 bytes, defined as [D3D12_RAYTRACING_AABB_BYTE_ALIGNMENT](#). The stride can be 0.

Requirements

Header	
d3d12.h	

D3D12_RAYTRACING_GEOMETRY_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	d3d12.h

D3D12_RAYTRACING_GEOMETRY_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies flags for raytracing geometry in a [D3D12_RAYTRACING_GEOMETRY_DESC](#) structure.

Syntax

```
typedef enum D3D12_RAYTRACING_GEOMETRY_FLAGS {  
    D3D12_RAYTRACING_GEOMETRY_FLAG_NONE,  
    D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE,  
    D3D12_RAYTRACING_GEOMETRY_FLAG_NO_DUPLICATE_ANYHIT_INVOCATION  
};
```

Constants

D3D12_RAYTRACING_GEOMETRY_FLAG_NONE	No options specified.
D3D12_RAYTRACING_GEOMETRY_FLAG_OPAQUE	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_AN YHIT_INVOCATION	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

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

D3D12_RAYTRACING_GEOMETRY_TRIANGLES_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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.

`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

<code>Header</code>	<code>d3d12.h</code>

D3D12_RAYTRACING_GEOMETRY_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_RAYTRACING_GEOMETRY_TYPE {
    D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES,
    D3D12_RAYTRACING_GEOMETRY_TYPE_PROCEDURAL_PRIMITIVE_AABBS
};
```

Constants

D3D12_RAYTRACING_GEOMETRY_TYPE_TRIANGLES	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

Header	d3d12.h

D3D12_RAYTRACING_INSTANCE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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 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

Header	
d3d12.h	

D3D12_RAYTRACING_INSTANCE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Flags for a raytracing acceleration structure instance. These flags can be used to override [D3D12_RAYTRACING_GEOMETRY_FLAGS](#) for individual instances.

Syntax

```
typedef enum D3D12_RAYTRACING_INSTANCE_FLAGS {  
    D3D12_RAYTRACING_INSTANCE_FLAG_NONE,  
    D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE,  
    D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_FRONT_COUNTERCLOCKWISE,  
    D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_OPAQUE,  
    D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE  
} ;
```

Constants

D3D12_RAYTRACING_INSTANCE_FLAG_NONE	No options specified.
D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_CULL_DISABLE	Disables front/back face culling for this instance. The Ray flags <code>RAY_FLAG_CULL_BACK_FACING_TRIANGLES</code> and <code>RAY_FLAG_CULL_FRONT_FACING_TRIANGLES</code> will have no effect on this instance.
D3D12_RAYTRACING_INSTANCE_FLAG_TRIANGLE_FRONT_COUNTERCLOCKWISE	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	<p>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.</p> <p>This flag is mutually exclusive to the D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE flag.</p>
D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_NON_OPAQUE	<p>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.</p> <p>This flag is mutually exclusive to the D3D12_RAYTRACING_INSTANCE_FLAG_FORCE_OPAQUE flag.</p>

Requirements

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

D3D12_RAYTRACING_PIPELINE_CONFIG structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A state subobject that represents a raytracing pipeline configuration.

Syntax

```
typedef struct D3D12_RAYTRACING_PIPELINE_CONFIG {
    UINT MaxTraceRecursionDepth;
} D3D12_RAYTRACING_PIPELINE_CONFIG;
```

Members

`MaxTraceRecursionDepth`

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 they must all match in content. There is 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

Header	
d3d12.h	

D3D12_RAYTRACING_SHADER_CONFIG structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A state subobject that represents a shader configuration.

Syntax

```
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

Header	
d3d12.h	

D3D12_RAYTRACING_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the level of ray tracing support on the graphics device.

Syntax

```
typedef enum D3D12_RAYTRACING_TIER {  
    D3D12_RAYTRACING_TIER_NOT_SUPPORTED,  
    D3D12_RAYTRACING_TIER_1_0,  
    D3D12_RAYTRACING_TIER_1_1  
} ;
```

Constants

D3D12_RAYTRACING_TIER_NOT_SUPPORTED	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	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_D3D12_OPTIONS5](#) to [ID3D12Device::CheckFeatureSupport](#) to retrieve a [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) struct.

Requirements

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

D3D12_RENDER_PASS_BEGINNING_ACCESS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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;
```

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.

Requirements

Header	d3d12.h

See also

[Rendering](#)

D3D12_RENDER_PASS_BEGINNING_ACCESS_CLEAR_PARAMETERS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	
d3d12.h	

See also

[Rendering](#)

D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of access that an application is given to the specified resource(s) at the transition into a render pass.

Syntax

```
typedef enum D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE {
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_DISCARD,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_PRESERVE,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_CLEAR,
    D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_NO_ACCESS
} ;
```

Constants

D3D12_RENDER_PASS_BEGINNING_ACCESS_TYPE_DISCARD	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

Header	
	d3d12.h

See also

[Rendering](#)

D3D12_RENDER_PASS_DEPTH_STENCIL_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	
d3d12.h	

See also

Rendering

D3D12_RENDER_PASS_ENDING_ACCESS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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;
```

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.

Requirements

Header	d3d12.h

See also

[Rendering](#)

D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_PARAMETERS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

Rendering

D3D12_RENDER_PASS_ENDING_ACCESS_RESOLVE_SUBRESOURCE_PARAMETERS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Rendering](#)

D3D12_RENDER_PASS_ENDING_ACCESS_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of access that an application is given to the specified resource(s) at the transition out of a render pass.

Syntax

```
typedef enum D3D12_RENDER_PASS_ENDING_ACCESS_TYPE {  
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_DISCARD,  
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_PRESERVE,  
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_RESOLVE,  
    D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_NO_ACCESS  
} ;
```

Constants

D3D12_RENDER_PASS_ENDING_ACCESS_TYPE_DISCARD	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 happenwhile 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

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

See also

[Rendering](#)

D3D12_RENDER_PASS_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the nature of the render pass; for example, whether it is a suspending or a resuming render pass.

Syntax

```
typedef enum D3D12_RENDER_PASS_FLAGS {  
    D3D12_RENDER_PASS_FLAG_NONE,  
    D3D12_RENDER_PASS_FLAG_ALLOW_UAV_WRITES,  
    D3D12_RENDER_PASS_FLAG_SUSPENDING_PASS,  
    D3D12_RENDER_PASS_FLAG_RESUMING_PASS  
} ;
```

Constants

D3D12_RENDER_PASS_FLAG_NONE	Indicates that the render pass has no special requirements.
D3D12_RENDER_PASS_FLAG_ALLOW_UAV_WRITES	Indicates that writes to unordered access view(s) should be allowed during the render pass.
D3D12_RENDER_PASS_FLAG_SUSPENDING_PASS	Indicates that this is a suspending render pass.
D3D12_RENDER_PASS_FLAG_RESUMING_PASS	Indicates that this is a resuming render pass.

Requirements

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

See also

[Rendering](#)

D3D12_RENDER_PASS_RENDER_TARGET_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Header	d3d12.h

See also

[Rendering](#)

D3D12_RENDER_PASS_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the level of support for render passes on a graphics device.

Syntax

```
typedef enum D3D12_RENDER_PASS_TIER {
    D3D12_RENDER_PASS_TIER_0,
    D3D12_RENDER_PASS_TIER_1,
    D3D12_RENDER_PASS_TIER_2
} ;
```

Constants

D3D12_RENDER_PASS_TIER_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 at this level of support.
D3D12_RENDER_PASS_TIER_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	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_D3D12_OPTIONS5](#) to [ID3D12Device::CheckFeatureSupport](#) to retrieve a [D3D12_FEATURE_DATA_D3D12_OPTIONS5](#) struct.

Requirements

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

See also

[Rendering](#)

D3D12_RENDER_TARGET_BLEND_DESC structure

4/28/2020 • 2 minutes to read • [Edit Online](#)

Describes the blend state for a render target.

Syntax

```
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.

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

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

See also

[Core Structures](#)

D3D12_RENDER_TARGET_VIEW_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_RESIDENCY_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Used with the EnqueueMakeResident function to choose how residency operations proceed when the memory budget is exceeded.

Syntax

```
typedef enum D3D12_RESIDENCY_FLAGS {
    D3D12_RESIDENCY_FLAG_NONE,
    D3D12_RESIDENCY_FLAG_DENY_OVERBUDGET
} ;
```

Constants

D3D12_RESIDENCY_FLAG_NONE	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	Specifies that the EnqueueMakeResident function should return E_OUTOFMEMORY when the residency operation would exceed the application's current memory budget.

Requirements

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_RESIDENCY_PRIORITY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_RESIDENCY_PRIORITY {  
    D3D12_RESIDENCY_PRIORITY_MINIMUM,  
    D3D12_RESIDENCY_PRIORITY_LOW,  
    D3D12_RESIDENCY_PRIORITY_NORMAL,  
    D3D12_RESIDENCY_PRIORITY_HIGH,  
    D3D12_RESIDENCY_PRIORITY_MAXIMUM  
};
```

Constants

D3D12_RESIDENCY_PRIORITY_MINIMUM	Indicates a minimum priority.
D3D12_RESIDENCY_PRIORITY_LOW	Indicates a low priority.
D3D12_RESIDENCY_PRIORITY_NORMAL	Indicates a normal, medium, priority.
D3D12_RESIDENCY_PRIORITY_HIGH	Indicates a high priority. Applications are discouraged from using priorities greater than this. For more information see ID3D12Device1::SetResidencyPriority .
D3D12_RESIDENCY_PRIORITY_MAXIMUM	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_RESOLVE_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a resolve operation.

Syntax

```
typedef enum D3D12_RESOLVE_MODE {  
    D3D12_RESOLVE_MODE_DECOMPRESS,  
    D3D12_RESOLVE_MODE_MIN,  
    D3D12_RESOLVE_MODE_MAX,  
    D3D12_RESOLVE_MODE_AVERAGE,  
    D3D12_RESOLVE_MODE_ENCODE_SAMPLER_FEEDBACK,  
    D3D12_RESOLVE_MODE_DECODE_SAMPLER_FEEDBACK  
} ;
```

Constants

D3D12_RESOLVE_MODE_DECOMPRESS	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	Resolves the source samples to their minimum value. It can be used with any render target or depth stencil format.
D3D12_RESOLVE_MODE_MAX	Resolves the source samples to their maximum value. It can be used with any render target or depth stencil format.
D3D12_RESOLVE_MODE_AVERAGE	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_RESOURCE_ALIASING_BARRIER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_RESOURCE_ALLOCATION_INFO structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes parameters needed to allocate resources.

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[CD3DX12_RESOURCE_ALLOCATION_INFO](#)

[Core structures](#)

D3D12_RESOURCE_ALLOCATION_INFO1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes parameters needed to allocate resources, including offset.

Syntax

```
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

Header	d3d12.h

See also

[Core structures](#)

D3D12_RESOURCE_BARRIER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a resource barrier (transition in resource use).

Syntax

```
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.

UAV

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

Header	d3d12.h

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_RESOURCE_BARRIER_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Flags for setting split resource barriers.

Syntax

```
typedef enum D3D12_RESOURCE_BARRIER_FLAGS {
    D3D12_RESOURCE_BARRIER_FLAG_NONE,
    D3D12_RESOURCE_BARRIER_FLAG_BEGIN_ONLY,
    D3D12_RESOURCE_BARRIER_FLAG_END_ONLY
} ;
```

Constants

D3D12_RESOURCE_BARRIER_FLAG_NONE	No flags.
D3D12_RESOURCE_BARRIER_FLAG_BEGIN_ONLY	This starts a barrier transition in a new state, putting a resource in a temporary no-access condition.
D3D12_RESOURCE_BARRIER_FLAG_END_ONLY	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

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

See also

[Core Enumerations](#)

[ResourceBarrier](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_RESOURCE_BARRIER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a type of resource barrier (transition in resource use) description.

Syntax

```
typedef enum D3D12_RESOURCE_BARRIER_TYPE {
    D3D12_RESOURCE_BARRIER_TYPE_TRANSITION,
    D3D12_RESOURCE_BARRIER_TYPE_ALIASING,
    D3D12_RESOURCE_BARRIER_TYPE_UAV
} ;
```

Constants

D3D12_RESOURCE_BARRIER_TYPE_TRANSITION	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

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

See also

[Core Enumerations](#)

D3D12_RESOURCE_BINDING_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the tier of resource binding being used.

Syntax

```
typedef enum D3D12_RESOURCE_BINDING_TIER {  
    D3D12_RESOURCE_BINDING_TIER_1,  
    D3D12_RESOURCE_BINDING_TIER_2,  
    D3D12_RESOURCE_BINDING_TIER_3  
} ;
```

Constants

D3D12_RESOURCE_BINDING_TIER_1	Tier 1. See Hardware Tiers .
D3D12_RESOURCE_BINDING_TIER_2	Tier 2. See Hardware Tiers .
D3D12_RESOURCE_BINDING_TIER_3	Tier 3. See Hardware Tiers .

Remarks

This enum is used by the [D3D12_FEATURE_DATA_D3D12_OPTIONS](#) structure.

Requirements

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

See also

[Core Enumerations](#)

[Hardware Tiers](#)

D3D12_RESOURCE_DESC structure

5/27/2020 • 4 minutes to read • [Edit Online](#)

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

Syntax

```
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_TEXTURE2D_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

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

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Structures](#)

[D3D12_HEAP_FLAGS](#)

D3D12_RESOURCE_DIMENSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the type of resource being used.

Syntax

```
typedef enum D3D12_RESOURCE_DIMENSION {  
    D3D12_RESOURCE_DIMENSION_UNKNOWN,  
    D3D12_RESOURCE_DIMENSION_BUFFER,  
    D3D12_RESOURCE_DIMENSION_TEXTURE1D,  
    D3D12_RESOURCE_DIMENSION_TEXTURE2D,  
    D3D12_RESOURCE_DIMENSION_TEXTURE3D  
} ;
```

Constants

D3D12_RESOURCE_DIMENSION_UNKNOWN	Resource is of unknown type.
D3D12_RESOURCE_DIMENSION_BUFFER	Resource is a buffer.
D3D12_RESOURCE_DIMENSION_TEXTURE1D	Resource is a 1D texture.
D3D12_RESOURCE_DIMENSION_TEXTURE2D	Resource is a 2D texture.
D3D12_RESOURCE_DIMENSION_TEXTURE3D	Resource is a 3D texture.

Remarks

This enum is used by the [D3D12_RESOURCE_DESC](#) structure.

Requirements

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

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Enumerations](#)

D3D12_RESOURCE_FLAGS enumeration

5/27/2020 • 4 minutes to read • [Edit Online](#)

Specifies options for working with resources.

Syntax

```
typedef enum D3D12_RESOURCE_FLAGS {
    D3D12_RESOURCE_FLAG_NONE,
    D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET,
    D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL,
    D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS,
    D3D12_RESOURCE_FLAG_DENY_SHADER_RESOURCE,
    D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER,
    D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS,
    D3D12_RESOURCE_FLAG_VIDEO_DECODE_REFERENCE_ONLY
} ;
```

Constants

D3D12_RESOURCE_FLAG_NONE	No options are specified.

D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET

Allows a render target view to be created for the resource, as well as 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. Applications 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.
- Cannot 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.
- Cannot be used with 4KB alignment, 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

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. Applications 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.
- Cannot be used with D3D12_RESOURCE_DIMENSION_BUFFER, 4KB alignment, D3D12_RESOURCE_FLAG_ALLOW_RENDER_TARGET, D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS, 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.

<p>D3D12_RESOURCE_FLAG_ALLOW_UNORDERED_ACCESS</p>	<p>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, it may be worth having two textures around and copying between them. One texture would have this flag, while the other wouldn't. Applications should avoid setting this flag when unordered access operations will never occur.</p> <p>The following restrictions and interactions apply:</p> <ul style="list-style-type: none"> 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. Cannot 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. Cannot be used with MSAA textures.
<p>D3D12_RESOURCE_FLAG_DENY_SHADER_RESOURCE</p>	<p>Disallows a shader resource view to be created for the resource, as well as disables the resource to transition into the state of D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE or D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE. Some adapter architectures experience increased bandwidth for depth stencil textures when shader resource views are precluded. If a texture is rarely used for shader resource, it may be worth having two textures around and copying between them. One texture would have this flag and the other wouldn't. Applications should set this flag when depth stencil textures will never be used from shader resource views.</p> <p>The following restrictions and interactions apply:</p> <ul style="list-style-type: none"> Must be used with D3D12_RESOURCE_FLAG_ALLOW_DEPTH_STENCIL.

D3D12_RESOURCE_FLAG_ALLOW_CROSS_ADAPTER	<p>Allows the resource to be used for cross-adapter data, as well as the same features enabled by ALLOW_SIMULTANEOUS_ACCESS. Cross adapter resources commonly preclude techniques that reduce effective texture bandwidth during usage, and some adapter architectures may require different caching behavior. Applications should avoid setting this flag when the resource data will never be used with another adapter.</p> <p>The following restrictions and interactions apply:</p> <ul style="list-style-type: none"> • Must be used with heaps that have D3D12_HEAP_FLAG_SHARED_CROSS_ADAPTER. • Cannot be used with heaps that have D3D12_HEAP_FLAG_ALLOW_DISPLAY.
D3D12_RESOURCE_FLAG_ALLOW_SIMULTANEOUS_ACCESS	<p>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).</p> <p>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 cannot leverage techniques to reduce effective texture bandwidth during usage.</p> <p>However, applications 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 prevents any compression being used with a resource.</p> <p>These restrictions and interactions apply.</p> <ul style="list-style-type: none"> - 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_FLAG_ALLOW_DEPTH_STENCIL.

D3D12_RESOURCE_FLAG_VIDEO_DECODE_REFERENCE_ONLY	<p>This resource may only be used as a decode reference frame. It may only be written to or read by the video decode operation.</p> <p>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 so, the application must allocate reference frames with the new D3D12_RESOURCE_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.</p>
---	---

Remarks

This enum is used by the **Flags** member of the [D3D12_RESOURCE_DESC](#) structure.

Requirements

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

See also

[Core Enumerations](#)

D3D12_RESOURCE_HEAP_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies which resource heap tier the hardware and driver support.

Syntax

```
typedef enum D3D12_RESOURCE_HEAP_TIER {
    D3D12_RESOURCE_HEAP_TIER_1,
    D3D12_RESOURCE_HEAP_TIER_2
} ;
```

Constants

D3D12_RESOURCE_HEAP_TIER_1	<p>Indicates that heaps can only support resources from a single resource category.</p> <p>For the list of resource categories, see Remarks.</p> <p>In tier 1, these resource categories are mutually exclusive and cannot be used with the same heap.</p> <p>The resource category must be declared when creating a heap, using the correct D3D12_HEAP_FLAGS enumeration constant.</p> <p>Applications cannot create heaps with flags that allow all three categories.</p>
D3D12_RESOURCE_HEAP_TIER_2	<p>Indicates that heaps can support resources from all three categories.</p> <p>For the list of resource categories, see Remarks.</p> <p>In tier 2, these resource categories can be mixed within the same heap.</p> <p>Applications may create heaps with flags that allow all three categories; but are not required to do so.</p> <p>Applications may be written to support tier 1 and seamlessly run on tier 2.</p>

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

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

See also

[Core Enumerations](#)

D3D12_RESOURCE_STATES enumeration

5/27/2020 • 5 minutes to read • [Edit Online](#)

Defines constants that specify the state of a resource regarding how the resource is being used.

Syntax

```
typedef enum D3D12_RESOURCE_STATES {  
    D3D12_RESOURCE_STATE_COMMON,  
    D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER,  
    D3D12_RESOURCE_STATE_INDEX_BUFFER,  
    D3D12_RESOURCE_STATE_RENDER_TARGET,  
    D3D12_RESOURCE_STATE_UNORDERED_ACCESS,  
    D3D12_RESOURCE_STATE_DEPTH_WRITE,  
    D3D12_RESOURCE_STATE_DEPTH_READ,  
    D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE,  
    D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE,  
    D3D12_RESOURCE_STATE_STREAM_OUT,  
    D3D12_RESOURCE_STATE_INDIRECT_ARGUMENT,  
    D3D12_RESOURCE_STATE_COPY_DEST,  
    D3D12_RESOURCE_STATE_COPY_SOURCE,  
    D3D12_RESOURCE_STATE_RESOLVE_DEST,  
    D3D12_RESOURCE_STATE_RESOLVE_SOURCE,  
    D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE,  
    D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE,  
    D3D12_RESOURCE_STATE_GENERIC_READ,  
    D3D12_RESOURCE_STATE_PRESENT,  
    D3D12_RESOURCE_STATE_PREDICATION,  
    D3D12_RESOURCE_STATE_VIDEO_DECODE_READ,  
    D3D12_RESOURCE_STATE_VIDEO_DECODE_WRITE,  
    D3D12_RESOURCE_STATE_VIDEO_PROCESS_READ,  
    D3D12_RESOURCE_STATE_VIDEO_PROCESS_WRITE,  
    D3D12_RESOURCE_STATE_VIDEO_ENCODE_READ,  
    D3D12_RESOURCE_STATE_VIDEO_ENCODE_WRITE  
};
```

Constants

D3D12_RESOURCE_STATE_COMMON

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 does not 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	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	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	<p>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.</p> <p>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.</p>
D3D12_RESOURCE_STATE_UNORDERED_ACCESS	<p>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.</p>
D3D12_RESOURCE_STATE_DEPTH_WRITE	<p>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).</p>
D3D12_RESOURCE_STATE_DEPTH_READ	<p>DEPTH_READ is a state which can be combined with other states. It should be used when the subresource is in a read-only depth stencil view, or when the <i>DepthEnable</i> parameter of D3D12_DEPTH_STENCIL_DESC is false. 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.</p>
D3D12_RESOURCE_STATE_NON_PIXEL_SHADER_RESOURCE	<p>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.</p>
D3D12_RESOURCE_STATE_PIXEL_SHADER_RESOURCE	<p>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.</p>

D3D12_RESOURCE_STATE_STREAM_OUT	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	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	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	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	The resource is used as the destination in a resolve operation.
D3D12_RESOURCE_STATE_RESOLVE_SOURCE	The resource is used as the source in a resolve operation.
D3D12_RESOURCE_STATE_RAYTRACING_ACCELERATION_STRUCTURE	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.
D3D12_RESOURCE_STATE_SHADING_RATE_SOURCE	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_PRESENT	Synonymous with D3D12_RESOURCE_STATE_COMMON.
D3D12_RESOURCE_STATE_PREDICATION	The resource is used for Predication .
D3D12_RESOURCE_STATE_VIDEO_DECODE_READ	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	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	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	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	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	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

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

See also

[Core Enumerations](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_RESOURCE_TRANSITION_BARRIER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the transition of subresources between different usages.

Syntax

```
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

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

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_RESOURCE_UAV_BARRIER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

[Using Resource Barriers to Synchronize Resource States in Direct3D 12](#)

D3D12_ROOT_CONSTANTS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

<code>Header</code>	<code>d3d12.h</code>

See also

[CD3DX12_ROOT_CONSTANTS](#)

[Core Structures](#)

[Creating a Root Signature](#)

[Using constants directly in the root signature](#)

D3D12_ROOT_DESCRIPTOR structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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 **SlotType** 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

Header	d3d12.h

See also

[CD3DX12_ROOT_DESCRIPTOR](#)

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR1](#)

[Using descriptors directly in the root signature](#)

D3D12_ROOT_DESCRIPTOR_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the volatility of the data referenced by descriptors in a Root Signature 1.1 description, which can enable some driver optimizations.

Syntax

```
typedef enum D3D12_ROOT_DESCRIPTOR_FLAGS {
    D3D12_ROOT_DESCRIPTOR_FLAG_NONE,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_VOLATILE,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE,
    D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC
} ;
```

Constants

D3D12_ROOT_DESCRIPTOR_FLAG_NONE	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	Data is volatile. Equivalent to Root Signature Version 1.0.
D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC_WHILE_SET_AT_EXECUTE	Data is static while set at execute.
D3D12_ROOT_DESCRIPTOR_FLAG_DATA_STATIC	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

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

See also

[Core Enumerations](#)

[Root Signature Version 1.1](#)

D3D12_ROOT_DESCRIPTOR_TABLE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the root signature 1.0 layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.

Syntax

```
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 **SlotType** member to [D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE](#).

Requirements

Header	d3d12.h

See also

[CD3DX12_ROOT_DESCRIPTOR_TABLE](#)

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR_TABLE1](#)

D3D12_ROOT_DESCRIPTOR_TABLE1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the root signature 1.1 layout of a descriptor table as a collection of descriptor ranges that appear one after the other in a descriptor heap.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR_TABLE](#)

[Root Signature Version 1.1](#)

D3D12_ROOT_DESCRIPTOR1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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 `SlotType` 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

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

See also

[Core Structures](#)

[D3D12_ROOT_DESCRIPTOR](#)

[Root Signature Version 1.1](#)

D3D12_ROOT_PARAMETER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the slot of a root signature version 1.0.

Syntax

```
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

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

See also

[CD3DX12_ROOT_PARAMETER](#)

[Core Structures](#)

[Creating a Root Signature](#)

[D3D12_ROOT_PARAMETER1](#)

D3D12_ROOT_PARAMETER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of root signature slot.

Syntax

```
typedef enum D3D12_ROOT_PARAMETER_TYPE {
    D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE,
    D3D12_ROOT_PARAMETER_TYPE_32BIT_CONSTANTS,
    D3D12_ROOT_PARAMETER_TYPE_CBV,
    D3D12_ROOT_PARAMETER_TYPE_SRV,
    D3D12_ROOT_PARAMETER_TYPE_UAV
} ;
```

Constants

D3D12_ROOT_PARAMETER_TYPE_DESCRIPTOR_TABLE	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

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

See also

[Core Enumerations](#)

[Creating a Root Signature](#)

D3D12_ROOT_PARAMETER1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the slot of a root signature version 1.1.

Syntax

```
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

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

See also

[Core Structures](#)

[D3D12_ROOT_PARAMETER](#)

[Root Signature Version 1.1](#)

D3D12_ROOT_SIGNATURE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the layout of a root signature version 1.0.

Syntax

```
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

Header	d3d12.h

See also

[CD3DX12_ROOT_SIGNATURE_DESC](#)

[Core Structures](#)

[Creating a Root Signature](#)

[D3D12_ROOT_PARAMETER_TYPE](#)

[D3D12_ROOT_SIGNATURE_DESC1](#)

[Using constants directly in the root signature](#)

[Using descriptors directly in the root signature](#)

D3D12_ROOT_SIGNATURE_DESC1 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the layout of a root signature version 1.1.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

[D3D12_ROOT_SIGNATURE_DESC](#)

Root Signature Version 1.1

D3D12_ROOT_SIGNATURE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies options for root signature layout.

Syntax

```
typedef enum D3D12_ROOT_SIGNATURE_FLAGS {
    D3D12_ROOT_SIGNATURE_FLAG_NONE,
    D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_VERTEX_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_HULL_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_DOMAIN_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_GEOMETRY_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_PIXEL_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_ALLOW_STREAM_OUTPUT,
    D3D12_ROOT_SIGNATURE_FLAG_LOCAL_ROOT_SIGNATURE,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_AMPLIFICATION_SHADER_ROOT_ACCESS,
    D3D12_ROOT_SIGNATURE_FLAG_DENY_MESH_SHADER_ROOT_ACCESS
} ;
```

Constants

D3D12_ROOT_SIGNATURE_FLAG_NONE	Indicates default behavior.
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_INPUT_ASSEMBLER_INPUT_LAYOUT	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	Denies the vertex shader access to the root signature.
D3D12_ROOT_SIGNATURE_FLAG_DENY_HULL_SHADER_ROOT_ACCESS	Denies the hull shader access to the root signature.
D3D12_ROOT_SIGNATURE_FLAG_DENY_DOMAIN_SHADER_ROOT_ACCESS	Denies the domain shader access to the root signature.
D3D12_ROOT_SIGNATURE_FLAG_DENY_GEOMETRY_SHADER_ROOT_ACCESS	Denies the geometry shader access to the root signature.
D3D12_ROOT_SIGNATURE_FLAG_DENY_PIXEL_SHADER_ROOT_ACCESS	Denies the pixel shader access to the root signature.
D3D12_ROOT_SIGNATURE_FLAG_ALLOW_STREAM_OUTPUT	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	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.
--	---

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

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

See also

[Core Enumerations](#)

[Creating a Root Signature](#)

[D3D12_ROOT_SIGNATURE_DESC](#)

D3D12_RT_FORMAT_ARRAY structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Wraps an array of render target formats.

Syntax

```
struct D3D12_RT_FORMAT_ARRAY {  
    DXGI_FORMAT RTFormats[8];  
    UINT        NumRenderTargets;  
};
```

Members

RTFormats

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

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

See also

[Core Structures](#)

D3D12_RTV_DIMENSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the type of resource to view as a render target.

Syntax

```
typedef enum D3D12_RTV_DIMENSION {
    D3D12_RTV_DIMENSION_UNKNOWN,
    D3D12_RTV_DIMENSION_BUFFER,
    D3D12_RTV_DIMENSION_TEXTURE1D,
    D3D12_RTV_DIMENSION_TEXTURE1DARRAY,
    D3D12_RTV_DIMENSION_TEXTURE2D,
    D3D12_RTV_DIMENSION_TEXTURE2DARRAY,
    D3D12_RTV_DIMENSION_TEXTURE2DMS,
    D3D12_RTV_DIMENSION_TEXTURE2DMSARRAY,
    D3D12_RTV_DIMENSION_TEXTURE3D
} ;
```

Constants

D3D12_RTV_DIMENSION_UNKNOWN	Do not use this value, as it will cause ID3D12Device::CreateRenderTargetView to fail.
D3D12_RTV_DIMENSION_BUFFER	The resource will be accessed as a buffer.
D3D12_RTV_DIMENSION_TEXTURE1D	The resource will be accessed as a 1D texture.
D3D12_RTV_DIMENSION_TEXTURE1DARRAY	The resource will be accessed as an array of 1D textures.
D3D12_RTV_DIMENSION_TEXTURE2D	The resource will be accessed as a 2D texture.
D3D12_RTV_DIMENSION_TEXTURE2DARRAY	The resource will be accessed as an array of 2D textures.
D3D12_RTV_DIMENSION_TEXTURE2DMS	The resource will be accessed as a 2D texture with multisampling.
D3D12_RTV_DIMENSION_TEXTURE2DMSARRAY	The resource will be accessed as an array of 2D textures with multisampling.
D3D12_RTV_DIMENSION_TEXTURE3D	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

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

See also

[Core Enumerations](#)

D3D12_SAMPLE_POSITION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_SAMPLER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a sampler state.

Syntax

```
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

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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

The opaque driver versioning data.

Requirements

Header	
d3d12.h	

D3D12_SERIALIZED_DATA_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of serialized data. Use a value from this enumeration when calling [ID3D12Device5::CheckDriverMatchingIdentifier](#).

Syntax

```
typedef enum D3D12_SERIALIZED_DATA_TYPE {
    D3D12_SERIALIZED_DATA_RAYTRACING_ACCELERATION_STRUCTURE
} ;
```

Constants

D3D12_SERIALIZED_DATA_RAYTRACING_ACCELERATION_STRUCTURE	The serialized data is a raytracing acceleration structure.
---	---

Requirements

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

D3D12_SERIALIZED_RAYTRACING_ACCELERATION_STRUCTURE_HEADER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines the header for a serialized raytracing acceleration structure.

Syntax

```
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_DESC](#).

Requirements

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

D3D12_SHADER_BYTECODE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes shader data.

Syntax

```
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.

Requirements

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

See also

[CD3DX12_SHADER_BYTECODE](#)

[Core Structures](#)

D3D12_SHADER_CACHE_SUPPORT_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the level of support for shader caching in the current graphics driver.

Syntax

```
typedef enum D3D12_SHADER_CACHE_SUPPORT_FLAGS {  
    D3D12_SHADER_CACHE_SUPPORT_NONE,  
    D3D12_SHADER_CACHE_SUPPORT_SINGLE_PSO,  
    D3D12_SHADER_CACHE_SUPPORT_LIBRARY,  
    D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_INPROC_CACHE,  
    D3D12_SHADER_CACHE_SUPPORT_AUTOMATIC_DISK_CACHE  
} ;
```

Constants

D3D12_SHADER_CACHE_SUPPORT_NONE	Indicates that the driver does not support shader caching.
D3D12_SHADER_CACHE_SUPPORT_SINGLE_PSO	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	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	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	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

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

See also

[Core Structures](#)

D3D12_SHADER_COMPONENT_MAPPING enumeration

7/1/2020 • 2 minutes to read • [Edit Online](#)

Specifies how memory gets routed by a shader resource view (SRV).

Syntax

```
typedef enum D3D12_SHADER_COMPONENT_MAPPING {  
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_0,  
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_1,  
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_2,  
    D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3,  
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0,  
    D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1  
} ;
```

Constants

D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_0	Indicates return component 0 (red).
D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_1	Indicates return component 1 (green).
D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_2	Indicates return component 2 (blue).
D3D12_SHADER_COMPONENT_MAPPING_FROM_MEMORY_COMPONENT_3	Indicates return component 3 (alpha).
D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_0	Indicates forcing the resulting value to 0.
D3D12_SHADER_COMPONENT_MAPPING_FORCE_VALUE_1	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.

```
#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

Header	
d3d12.h	

See also

[Core enumerations](#)

D3D12_SHADER_MIN_PRECISION_SUPPORT enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes minimum precision support options for shaders in the current graphics driver.

Syntax

```
typedef enum D3D12_SHADER_MIN_PRECISION_SUPPORT {  
    D3D12_SHADER_MIN_PRECISION_SUPPORT_NONE,  
    D3D12_SHADER_MIN_PRECISION_SUPPORT_10_BIT,  
    D3D12_SHADER_MIN_PRECISION_SUPPORT_16_BIT  
} ;
```

Constants

D3D12_SHADER_MIN_PRECISION_SUPPORT_NONE	The driver supports only full 32-bit precision for all shader stages.
D3D12_SHADER_MIN_PRECISION_SUPPORT_10_BIT	The driver supports 10-bit precision.
D3D12_SHADER_MIN_PRECISION_SUPPORT_16_BIT	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

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

See also

[Core Enumerations](#)

D3D12_SHADER_RESOURCE_VIEW_DESC structure

7/1/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader-resource view (SRV).

Syntax

```
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

Header	d3d12.h

See also

[Core structures](#)

D3D12_SHADER_VISIBILITY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the shaders that can access the contents of a given root signature slot.

Syntax

```
typedef enum D3D12_SHADER_VISIBILITY {
    D3D12_SHADER_VISIBILITY_ALL,
    D3D12_SHADER_VISIBILITY_VERTEX,
    D3D12_SHADER_VISIBILITY_HULL,
    D3D12_SHADER_VISIBILITY_DOMAIN,
    D3D12_SHADER_VISIBILITY_GEOMETRY,
    D3D12_SHADER_VISIBILITY_PIXEL,
    D3D12_SHADER_VISIBILITY_AMPLIFICATION,
    D3D12_SHADER_VISIBILITY_MESH
} ;
```

Constants

D3D12_SHADER_VISIBILITY_ALL	Specifies that all shader stages can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_VERTEX	Specifies that the vertex shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_HULL	Specifies that the hull shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_DOMAIN	Specifies that the domain shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_GEOMETRY	Specifies that the geometry shader stage can access whatever is bound at the root signature slot.
D3D12_SHADER_VISIBILITY_PIXEL	Specifies that the pixel 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

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

See also

[Core Enumerations](#)

D3D12_SO_DECLARATION_ENTRY structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a vertex element in a vertex buffer in an output slot.

Syntax

```
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

Header	
	d3d12.h

See also

[Core Structures](#)

D3D12_SRV_DIMENSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the type of resource that will be viewed as a shader resource.

Syntax

```
typedef enum D3D12_SRV_DIMENSION {
    D3D12_SRV_DIMENSION_UNKNOWN,
    D3D12_SRV_DIMENSION_BUFFER,
    D3D12_SRV_DIMENSION_TEXTURE1D,
    D3D12_SRV_DIMENSION_TEXTURE1DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2D,
    D3D12_SRV_DIMENSION_TEXTURE2DARRAY,
    D3D12_SRV_DIMENSION_TEXTURE2DMS,
    D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY,
    D3D12_SRV_DIMENSION_TEXTURE3D,
    D3D12_SRV_DIMENSION_TEXTURECUBE,
    D3D12_SRV_DIMENSION_TEXTURECUBEARRAY,
    D3D12_SRV_DIMENSION_RAYTRACING_ACCELERATION_STRUCTURE
} ;
```

Constants

D3D12_SRV_DIMENSION_UNKNOWN	The type is unknown.
D3D12_SRV_DIMENSION_BUFFER	The resource is a buffer.
D3D12_SRV_DIMENSION_TEXTURE1D	The resource is a 1D texture.
D3D12_SRV_DIMENSION_TEXTURE1DARRAY	The resource is an array of 1D textures.
D3D12_SRV_DIMENSION_TEXTURE2D	The resource is a 2D texture.
D3D12_SRV_DIMENSION_TEXTURE2DARRAY	The resource is an array of 2D textures.
D3D12_SRV_DIMENSION_TEXTURE2DMS	The resource is a multisampling 2D texture.
D3D12_SRV_DIMENSION_TEXTURE2DMSARRAY	The resource is an array of multisampling 2D textures.
D3D12_SRV_DIMENSION_TEXTURE3D	The resource is a 3D texture.
D3D12_SRV_DIMENSION_TEXTURECUBE	The resource is a cube texture.
D3D12_SRV_DIMENSION_TEXTURECUBEARRAY	The resource is an array of cube textures.
D3D12_SRV_DIMENSION_RAYTRACING_ACCELERATION_STRUCTURE	The resource is a raytracing acceleration structure.

Remarks

These values are used by a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header	
d3d12.h	

See also

[Core Enumerations](#)

D3D12_STATE_OBJECT_CONFIG structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines general properties of a state object.

Syntax

```
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

Header	
d3d12.h	

D3D12_STATE_OBJECT_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Description of a state object. Pass a value of this structure type to [ID3D12Device5::CreateStateObject](#).

Syntax

```
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

Header	d3d12.h

D3D12_STATE_OBJECT_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies constraints for state objects. Use values from this enumeration in the [D3D12_STATE_OBJECT_CONFIG](#) structure.

Syntax

```
typedef enum D3D12_STATE_OBJECT_FLAGS {
    D3D12_STATE_OBJECT_FLAG_NONE,
    D3D12_STATE_OBJECT_FLAG_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITIONS,
    D3D12_STATE_OBJECT_FLAG_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS,
    D3D12_STATE_OBJECT_FLAG_ALLOW_STATE_OBJECT_ADDITIONS
} ;
```

Constants

D3D12_STATE_OBJECT_FLAG_NONE	No state object constraints.
D3D12_STATE_OBJECT_FLAG_ALLOW_LOCAL_DEPENDENCIES_ON_EXTERNAL_DEFINITIONS	<p>This flag applies to state objects of type collection only. Otherwise this flag is ignored.</p> <p>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.</p> <p>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.</p>

D3D12_STATE_OBJECT_FLAG_ALLOW_EXTERNAL_DEPENDENCIES_ON_LOCAL_DEFINITIONS	<p>This flag applies to state objects of type collection only. Otherwise this flag is ignored.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p>
--	--

Requirements

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

D3D12_STATE_OBJECT_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of a state object. Use with [D3D12_STATE_OBJECT_DESC](#).

Syntax

```
typedef enum D3D12_STATE_OBJECT_TYPE {
    D3D12_STATE_OBJECT_TYPE_COLLECTION,
    D3D12_STATE_OBJECT_TYPE_RAYTRACING_PIPELINE
} ;
```

Constants

D3D12_STATE_OBJECT_TYPE_COLLECTION	Collection state object.
D3D12_STATE_OBJECT_TYPE_RAYTRACING_PIPELINE	Raytracing pipeline state object.

Requirements

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

D3D12_STATE_SUBOBJECT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a subobject within a state object description. Use with [D3D12_STATE_OBJECT_DESC](#).

Syntax

```
typedef struct D3D12_STATE_SUBOBJECT {
    D3D12_STATE_SUBOBJECT_TYPE Type;
    const void               *pDesc;
} D3D12_STATE_SUBOBJECT;
```

Members

Type

The type of the state subobject.

pDesc

Pointer to state object description of the type specified in the *Type* parameter.

Requirements

Header	File
d3d12.h	

D3D12_STATE_SUBOBJECT_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

The type of a state subobject. Use with [D3D12_STATE_SUBOBJECT](#).

Syntax

```
typedef enum D3D12_STATE_SUBOBJECT_TYPE {
    D3D12_STATE_SUBOBJECT_TYPE_STATE_OBJECT_CONFIG,
    D3D12_STATE_SUBOBJECT_TYPE_GLOBAL_ROOT_SIGNATURE,
    D3D12_STATE_SUBOBJECT_TYPE_LOCAL_ROOT_SIGNATURE,
    D3D12_STATE_SUBOBJECT_TYPE_NODE_MASK,
    D3D12_STATE_SUBOBJECT_TYPE_DXIL_LIBRARY,
    D3D12_STATE_SUBOBJECT_TYPE_EXISTING_COLLECTION,
    D3D12_STATE_SUBOBJECT_TYPE_SUBOBJECT_TO_EXPORTS_ASSOCIATION,
    D3D12_STATE_SUBOBJECT_TYPE_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_SHADER_CONFIG,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG,
    D3D12_STATE_SUBOBJECT_TYPE_HIT_GROUP,
    D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG1,
    D3D12_STATE_SUBOBJECT_TYPE_MAX_VALID
} ;
```

Constants

D3D12_STATE_SUBOBJECT_TYPE_STATE_OBJECT_CONFIG	Subobject type is D3D12_STATE_OBJECT_CONFIG .
D3D12_STATE_SUBOBJECT_TYPE_GLOBAL_ROOT_SIGNATURE	Subobject type is D3D12_GLOBAL_ROOT_SIGNATURE .
D3D12_STATE_SUBOBJECT_TYPE_LOCAL_ROOT_SIGNATURE	Subobject type is D3D12_LOCAL_ROOT_SIGNATURE .
D3D12_STATE_SUBOBJECT_TYPE_NODE_MASK	Subobject type is D3D12_NODE_MASK .
D3D12_STATE_SUBOBJECT_TYPE_DXIL_LIBRARY	Subobject type is D3D12_DXIL_LIBRARY_DESC .
D3D12_STATE_SUBOBJECT_TYPE_EXISTING_COLLECTION	Subobject type is D3D12_EXISTING_COLLECTION_DESC .
D3D12_STATE_SUBOBJECT_TYPE_SUBOBJECT_TO_EXPORTS_ASSOCIATION	Subobject type is D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION .
D3D12_STATE_SUBOBJECT_TYPE_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION	Subobject type is D3D12_DXIL_SUBOBJECT_TO_EXPORTS_ASSOCIATION .
D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_SHADER_CONFIG	Subobject type is D3D12_RAYTRACING_SHADER_CONFIG .
D3D12_STATE_SUBOBJECT_TYPE_RAYTRACING_PIPELINE_CONFIG	Subobject type is D3D12_RAYTRACING_PIPELINE_CONFIG .
D3D12_STATE_SUBOBJECT_TYPE_HIT_GROUP	Subobject type is D3D12_HIT_GROUP_DESC .

D3D12_STATE_SUBOBJECT_TYPE_MAX_VALID	The maximum valid subobject type value.
--------------------------------------	---

Requirements

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

D3D12_STATIC_BORDER_COLOR enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the border color for a static sampler.

Syntax

```
typedef enum D3D12_STATIC_BORDER_COLOR {
    D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_BLACK,
    D3D12_STATIC_BORDER_COLOR_OPAQUE_WHITE
} ;
```

Constants

D3D12_STATIC_BORDER_COLOR_TRANSPARENT_BLACK	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

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

See also

[Core Enumerations](#)

D3D12_STATIC_SAMPLER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a static sampler.

Syntax

```
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:

```
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

Header	d3d12.h

See also

[CD3DX12_STATIC_SAMPLER_DESC](#)

[Core Structures](#)

D3D12_STENCIL_OP enumeration

6/25/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_STENCIL_OP {  
    D3D12_STENCIL_OP_KEEP,  
    D3D12_STENCIL_OP_ZERO,  
    D3D12_STENCIL_OP_REPLACE,  
    D3D12_STENCIL_OP_INCR_SAT,  
    D3D12_STENCIL_OP_DECR_SAT,  
    D3D12_STENCIL_OP_INVERT,  
    D3D12_STENCIL_OP_INCR,  
    D3D12_STENCIL_OP_DECR  
} ;
```

Constants

D3D12_STENCIL_OP_KEEP	Keep the existing stencil data.
D3D12_STENCIL_OP_ZERO	Set the stencil data to 0.
D3D12_STENCIL_OP_REPLACE	Set the stencil data to the reference value set by calling ID3D12GraphicsCommandList::OMSetStencilRef .
D3D12_STENCIL_OP_INCR_SAT	Increment the stencil value by 1, and clamp the result.
D3D12_STENCIL_OP_DECR_SAT	Decrement the stencil value by 1, and clamp the result.
D3D12_STENCIL_OP_INVERT	Invert the stencil data.
D3D12_STENCIL_OP_INCR	Increment the stencil value by 1, and wrap the result if necessary.
D3D12_STENCIL_OP_DECR	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

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

See also

[CD3DX12_DEPTH_STENCIL_DESC](#)

[Core enumerations](#)

D3D12_STREAM_OUTPUT_BUFFER_VIEW structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a stream output buffer.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_STREAM_OUTPUT_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a streaming output buffer.

Syntax

```
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

Header	File
d3d12.h	

See also

[Core Structures](#)

D3D12_SUBOBJECT_TO_EXPORTS_ASSOCIATION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

D3D12_SUBRESOURCE_DATA structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes subresource data.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_SUBRESOURCE_FOOTPRINT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header

d3d12.h

See also

CD3DX12_SUBRESOURCE_FOOTPRINT

Core Structures

D3D12_SUBRESOURCE_INFO structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes subresource data.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_SUBRESOURCE_RANGE_UINT64 structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a subresource memory range.

Syntax

```
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

Header	d3d12.h

See also

Core Structures

D3D12_SUBRESOURCE_TILING structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a tiled subresource volume.

Syntax

```
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

Header	d3d12.h

See also

[CD3DX12_SUBRESOURCE_TILING](#)

[Core Structures](#)

D3D12_TEX1D_ARRAY_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_ARRAY_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_ARRAY_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header	d3d12.h

See also

Core Structures

D3D12_TEX1D_ARRAY_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX1D_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a unordered-access 1D texture resource.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2D_ARRAY_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2D_ARRAY_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_TEX2D_ARRAY_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

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

See also

[Core Structures](#)

D3D12_TEX2D_ARRAY_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_TEX2D_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2D_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2D_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef struct D3D12_TEX2D_SRV {  
    UINT  MostDetailedMip;  
    UINT  MipLevels;  
    UINT  PlaneSlice;  
    FLOAT ResourceMinLODClamp;  
} D3D12_TEX2D_SRV;
```

Members

MostDetailedMip

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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2D_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a unordered-access 2D texture resource.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2DMS_ARRAY_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2DMS_ARRAY_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2DMS_ARRAY_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

<code>Header</code>	d3d12.h

See also

[Core Structures](#)

D3D12_TEX2DMS_DSV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header		d3d12.h

See also

[Core Structures](#)

D3D12_TEX2DMS_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_TEX2DMS_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_TEX3D_RTV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEX3D_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header

d3d12.h

See also

[Core Structures](#)

D3D12_TEX3D_UAV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a unordered-access 3D texture resource.

Syntax

```
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.

Remarks

Use this structure with a [D3D12_UNORDERED_ACCESS_VIEW_DESC](#) structure to view the resource as a 3D texture.

Requirements

Header	d3d12.h

See also

[Core Structures](#)

D3D12_TEXCUBE_ARRAY_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header	d3d12.h

See also

Core Structures

D3D12_TEXCUBE_SRV structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

A value to clamp sample LOD values to. For example, if you specify 2.0f for the clamp value, you ensure that no individual sample accesses a mip level less than 2.0f.

Remarks

This structure is one member of a shader-resource-view description, [D3D12_SHADER_RESOURCE_VIEW_DESC](#).

Requirements

Header

d3d12.h

See also

[Core Structures](#)

D3D12_TEXTURE_ADDRESS_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
typedef enum D3D12_TEXTURE_ADDRESS_MODE {  
    D3D12_TEXTURE_ADDRESS_MODE_WRAP,  
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR,  
    D3D12_TEXTURE_ADDRESS_MODE_CLAMP,  
    D3D12_TEXTURE_ADDRESS_MODE_BORDER,  
    D3D12_TEXTURE_ADDRESS_MODE_MIRROR_ONCE  
};
```

Constants

D3D12_TEXTURE_ADDRESS_MODE_WRAP	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	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	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	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	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

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

See also

[Core Enumerations](#)

D3D12_TEXTURE_COPY_LOCATION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

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

See also

[CD3DX12_TEXTURE_COPY_LOCATION](#)

[Core Structures](#)

[D3D12_PLACED_SUBRESOURCE_FOOTPRINT](#)

D3D12_TEXTURE_COPY_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies what type of texture copy is to take place.

Syntax

```
typedef enum D3D12_TEXTURE_COPY_TYPE {
    D3D12_TEXTURE_COPY_TYPE_SUBRESOURCE_INDEX,
    D3D12_TEXTURE_COPY_TYPE_PLACED_FOOTPRINT
} ;
```

Constants

D3D12_TEXTURE_COPY_TYPE_SUBRESOURCE_INDEX	Indicates a subresource, identified by an index, is to be copied.
D3D12_TEXTURE_COPY_TYPE_PLACED_FOOTPRINT	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

Header	d3d12.h

See also

[Core Enumerations](#)

D3D12_TEXTURE_LAYOUT enumeration

5/27/2020 • 3 minutes to read • [Edit Online](#)

Specifies texture layout options.

Syntax

```
typedef enum D3D12_TEXTURE_LAYOUT {  
    D3D12_TEXTURE_LAYOUT_UNKNOWN,  
    D3D12_TEXTURE_LAYOUT_ROW_MAJOR,  
    D3D12_TEXTURE_LAYOUT_64KB_UNDEFINED_SWIZZLE,  
    D3D12_TEXTURE_LAYOUT_64KB_STANDARD_SWIZZLE  
} ;
```

Constants

D3D12_TEXTURE_LAYOUT_UNKNOWN	<p>Indicates that the layout is unknown, and is likely adapter-dependent.</p> <p>During creation, the driver chooses the most efficient layout based on other resource properties, especially resource size and flags.</p> <p>Prefer this choice unless certain functionality is required from another texture layout.</p> <p>Zero-copy texture upload optimizations exist for UMA architectures; see ID3D12Resource::WriteToSubresource.</p>
D3D12_TEXTURE_LAYOUT_ROW_MAJOR	<p>Indicates that data for the texture is stored in row-major order (sometimes called "pitch-linear order").</p> <p>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.</p> <p>Padding may exist between rows and between depth or array slices to align collections of data.</p> <p>A stride is the distance in memory between rows, depth, or array slices; and it includes any padding.</p>
	<p>This texture layout enables sharing of the texture data between multiple adapters, when other layouts aren't available.</p> <p>Many restrictions apply, because this layout is generally not efficient for extensive usage:</p> <ul style="list-style-type: none">• 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

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	<p>Indicates that a default texture uses the standardized swizzle pattern.</p> <p>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.</p> <p>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.</p> <p>The restrictions for this layout are that the following aren't supported:</p> <ul style="list-style-type: none"> • 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	XYYX XYXY YYYY XXXX		
16bpp	XYYX XYXY YYYY XXX-	32bpp	XYYX XYXY YYYY XX--
64bpp	XYYX XYXY XXXY X---	128bpp	XYYX XYXY XXXY ----

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:

```
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):

```
UINT swizzledOffset = resourceBaseOffset +  
    _pdep_u32(xOffset, xBytesMask) +  
    _pdep_u32(yOffset, yBytesMask);
```

Requirements

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

See also

[CD3DX12_RESOURCE_DESC](#)

[Core Enumerations](#)

[UMA Optimizations: CPU Accessible Textures and Standard Swizzle](#)

D3D12_TILE_COPY_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies how to copy a tile.

Syntax

```
typedef enum D3D12_TILE_COPY_FLAGS {
    D3D12_TILE_COPY_FLAG_NONE,
    D3D12_TILE_COPY_FLAG_NO_HAZARD,
    D3D12_TILE_COPY_FLAG_LINEAR_BUFFER_TO_SWIZZLED_TILED_RESOURCE,
    D3D12_TILE_COPY_FLAG_SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER
} ;
```

Constants

D3D12_TILE_COPY_FLAG_NONE	No tile-copy flags are specified.
D3D12_TILE_COPY_FLAG_NO_HAZARD	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	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 <i>pBuffer</i> parameter and the destination with the <i>pTiledResource</i> parameter.
D3D12_TILE_COPY_FLAG_SWIZZLED_TILED_RESOURCE_TO_LINEAR_BUFFER	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 <i>pTiledResource</i> parameter and the destination with the <i>pBuffer</i> parameter.

Remarks

This enum is used by the [CopyTiles](#) method.

Requirements

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

See also

[Core Enumerations](#)

D3D12_TILE_MAPPING_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies how to perform a tile-mapping operation.

Syntax

```
typedef enum D3D12_TILE_MAPPING_FLAGS {  
    D3D12_TILE_MAPPING_FLAG_NONE,  
    D3D12_TILE_MAPPING_FLAG_NO_HAZARD  
} ;
```

Constants

D3D12_TILE_MAPPING_FLAG_NONE	No tile-mapping flags are specified.
D3D12_TILE_MAPPING_FLAG_NO_HAZARD	Unsupported, do not use.

Remarks

This enum is used by the following methods:

- [CopyTileMappings](#)
- [UpdateTileMappings](#)

Requirements

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

See also

[Core Enumerations](#)

D3D12_TILE_RANGE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a range of tile mappings.

Syntax

```
typedef enum D3D12_TILE_RANGE_FLAGS {  
    D3D12_TILE_RANGE_FLAG_NONE,  
    D3D12_TILE_RANGE_FLAG_NULL,  
    D3D12_TILE_RANGE_FLAG_SKIP,  
    D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE  
} ;
```

Constants

D3D12_TILE_RANGE_FLAG_NONE	No tile-mapping flags are specified.
D3D12_TILE_RANGE_FLAG_NULL	The tile range is NULL .
D3D12_TILE_RANGE_FLAG_SKIP	Skip the tile range.
D3D12_TILE_RANGE_FLAG_REUSE_SINGLE_TILE	Reuse a single tile in the tile range.

Remarks

Use these flags with [ID3D12CommandQueue::UpdateTileMappings](#).

Requirements

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

See also

[Core Enumerations](#)

D3D12_TILE_REGION_SIZE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the size of a tiled region.

Syntax

```
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. For more info, see [Tile and toast image sizes](#).

Height

The height of the tiled region, in tiles. Used for 2D and 3D textures. For more info, see [Tile and toast image sizes](#).

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

Header	d3d12.h

See also

[CD3DX12_TILE_REGION_SIZE](#)

[Core Structures](#)

D3D12_TILE_SHAPE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the shape of a tile by specifying its dimensions.

Syntax

```
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

Header	d3d12.h

See also

[CD3DX12_TILE_SHAPE](#)

[Core Structures](#)

D3D12_TILED_RESOURCE_COORDINATE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the coordinates of a tiled resource.

Syntax

```
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.

Remarks

This structure is used by the [CopyTiles](#), [CopyTileMappings](#) and [UpdateTileMappings](#) methods.

Requirements

Header	d3d12.h

See also

[CD3DX12_TILED_RESOURCE_COORDINATE](#)

[Core Structures](#)

D3D12_TILED_RESOURCES_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies the tier level at which tiled resources are supported.

Syntax

```
typedef enum D3D12_TILED_RESOURCES_TIER {  
    D3D12_TILED_RESOURCES_TIER_NOT_SUPPORTED,  
    D3D12_TILED_RESOURCES_TIER_1,  
    D3D12_TILED_RESOURCES_TIER_2,  
    D3D12_TILED_RESOURCES_TIER_3,  
    D3D12_TILED_RESOURCES_TIER_4  
} ;
```

Constants

D3D12_TILED_RESOURCES_TIER_NOT_SUPPORTED	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	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	<p>Indicates that a superset of Tier_1 functionality is supported, including this additional support:</p> <ul style="list-style-type: none"> 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. <p>Adapters that support feature level 12_0 all support TIER_2 or greater.</p>
D3D12_TILED_RESOURCES_TIER_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	

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

Header	
	d3d12.h

See also

[Core Enumerations](#)

D3D12_UAV_DIMENSION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Identifies unordered-access view options.

Syntax

```
typedef enum D3D12_UAV_DIMENSION {
    D3D12_UAV_DIMENSION_UNKNOWN,
    D3D12_UAV_DIMENSION_BUFFER,
    D3D12_UAV_DIMENSION_TEXTURE1D,
    D3D12_UAV_DIMENSION_TEXTURE1DARRAY,
    D3D12_UAV_DIMENSION_TEXTURE2D,
    D3D12_UAV_DIMENSION_TEXTURE2DARRAY,
    D3D12_UAV_DIMENSION_TEXTURE3D
};
```

Constants

D3D12_UAV_DIMENSION_UNKNOWN	The view type is unknown.
D3D12_UAV_DIMENSION_BUFFER	View the resource as a buffer.
D3D12_UAV_DIMENSION_TEXTURE1D	View the resource as a 1D texture.
D3D12_UAV_DIMENSION_TEXTURE1DARRAY	View the resource as a 1D texture array.
D3D12_UAV_DIMENSION_TEXTURE2D	View the resource as a 2D texture.
D3D12_UAV_DIMENSION_TEXTURE2DARRAY	View the resource as a 2D texture array.
D3D12_UAV_DIMENSION_TEXTURE3D	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

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

See also

[Core Enumerations](#)

D3D12_UNORDERED_ACCESS_VIEW_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the subresources from a resource that are accessible by using an unordered-access view.

Syntax

```
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_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.

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

Header	
	d3d12.h

See also

[Core Structures](#)

D3D12_VERSIONED_DEVICE_REMOVED_EXTENDED_DATA structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents versioned Device Removed Extended Data (DRED) data, so that debuggers and debugger extensions can access DRED data.

Syntax

```
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_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

Requirements

Header	File
	d3d12.h

See also

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

D3D12_VERSIONED_ROOT_SIGNATURE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Holds any version of a root signature description, and is designed to be used with serialization/deserialization functions.

Syntax

```
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_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).

Remarks

Use this structure with the following methods.

- [GetRootSignatureDescAtVersion](#)
- [GetUnconvertedRootSignatureDesc](#)
- [D3D12SerializeVersionedRootSignature](#)

Refer to the helper structure [CD3DX12_VERSIONED_ROOT_SIGNATURE_DESC](#).

Requirements

Header	d3d12.h

See also

[Core Structures](#)

Root Signature Version 1.1

D3D12_VERTEX_BUFFER_VIEW structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a vertex buffer view.

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12_VIEW_INSTANCE_LOCATION structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header		d3d12.h

See also

[Core Structures](#)

D3D12_VIEW_INSTANCING_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies parameters used during view instancing configuration.

Syntax

```
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

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

See also

[Core Structures](#)

D3D12_VIEW_INSTANCING_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies options for view instancing.

Syntax

```
typedef enum D3D12_VIEW_INSTANCING_FLAGS {  
    D3D12_VIEW_INSTANCING_FLAG_NONE,  
    D3D12_VIEW_INSTANCING_FLAG_ENABLE_VIEW_INSTANCE_MASKING  
} ;
```

Constants

D3D12_VIEW_INSTANCING_FLAG_NONE	Indicates a default view instancing configuration.
D3D12_VIEW_INSTANCING_FLAG_ENABLE_VIEW_INSTANCE_MASKING	Enables view instance masking.

Requirements

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

See also

[Core Enumerations](#)

[D3D12_VIEW_INSTANCING_DESC](#)

D3D12_VIEW_INSTANCING_TIER enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates the tier level at which view instancing is supported.

Syntax

```
typedef enum D3D12_VIEW_INSTANCING_TIER {  
    D3D12_VIEW_INSTANCING_TIER_NOT_SUPPORTED,  
    D3D12_VIEW_INSTANCING_TIER_1,  
    D3D12_VIEW_INSTANCING_TIER_2,  
    D3D12_VIEW_INSTANCING_TIER_3  
} ;
```

Constants

D3D12_VIEW_INSTANCING_TIER_NOT_SUPPORTED	View instancing is not supported.
D3D12_VIEW_INSTANCING_TIER_1	View instancing is supported by draw-call level looping only.
D3D12_VIEW_INSTANCING_TIER_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

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

Header

d3d12.h

See also

[Core Enumerations](#)

D3D12_VIEWPORT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the dimensions of a viewport.

Syntax

```
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

Header

d3d12.h

See also

[Core Structures](#)

D3D12_WRITEBUFFERIMMEDIATE_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the mode used by a **WriteBufferImmediate** operation.

Syntax

```
typedef enum D3D12_WRITEBUFFERIMMEDIATE_MODE {  
    D3D12_WRITEBUFFERIMMEDIATE_MODE_DEFAULT,  
    D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_IN,  
    D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_OUT  
} ;
```

Constants

D3D12_WRITEBUFFERIMMEDIATE_MODE_DEFAULT	The write operation behaves the same as normal copy-write operations.
D3D12_WRITEBUFFERIMMEDIATE_MODE_MARKER_IN	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	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

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

See also

[Core Enumerations](#)

[ID3D12GraphicsCommandList::WriteBufferImmediate](#)

D3D12_WRITEBUFFERIMMEDIATE_PARAMETER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the immediate value and destination address written using [ID3D12CommandList2::WriteBufferImmediate](#).

Syntax

```
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

Header	d3d12.h

See also

[Core Structures](#)

D3D12CreateDevice function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a device that represents the display adapter.

Syntax

```
HRESULT D3D12CreateDevice(
    IUnknown           *pAdapter,
    D3D_FEATURE_LEVEL MinimumFeatureLevel,
    REFIID             riid,
    void               **ppDevice
);
```

Parameters

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.

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](#).

ppDevice

Type: [void**](#)

A pointer to a memory block that receives a pointer to the device.

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](#).

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.

```
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.

```

ComPtr factory;
ThrowIfFailed(CreateDXGIFactory1(IID_PPV_ARGS(&factory)));

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

    ThrowIfFailed(D3D12CreateDevice(
        warpAdapter.Get(),
        D3D_FEATURE_LEVEL_11_0,
        IID_PPV_ARGS(&m_device)
    ));
}
else
{
    ComPtr 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Working Samples](#)

D3D12CreateRootSignatureDeserializer function

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT D3D12CreateRootSignatureDeserializer(
    LPCVOID pSrcData,
    SIZE_T SrcDataSizeInBytes,
    REFIID pRootSignatureDeserializerInterface,
    void    **ppRootSignatureDeserializer
);
```

Parameters

`pSrcData`

Type: `LPCVOID`

A pointer to the source data for the serialized root signature.

`SrcDataSizeInBytes`

Type: `SIZE_T`

The size, in bytes, of the block of memory that *pSrcData* points to.

`pRootSignatureDeserializerInterface`

Type: `REFIID`

The globally unique identifier ([GUID](#)) for the root signature deserializer interface. See remarks.

`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 deserialized data structure ([D3D12_ROOT_SIGNATURE_DESC](#)). `ID3D12RootSignatureDeserializer` just owns the lifetime of the memory for the deserialized 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Creating a Root Signature](#)

D3D12CreateVersionedRootSignatureDeserializer function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Generates an interface that can return the deserialized data structure, via [GetUnconvertedRootSignatureDesc](#).

Syntax

```
HRESULT D3D12CreateVersionedRootSignatureDeserializer(
    LPCVOID pSrcData,
    SIZE_T SrcDataSizeInBytes,
    REFIID pRootSignatureDeserializerInterface,
    void    **ppRootSignatureDeserializer
);
```

Parameters

`pSrcData`

Type: **LPCVOID**

A pointer to the source data for the serialized root signature.

`SrcDataSizeInBytes`

Type: **SIZE_T**

The size, in bytes, of the block of memory that *pSrcData* points to.

`pRootSignatureDeserializerInterface`

Type: **REFIID**

The globally unique identifier (GUID) for the root signature deserializer interface. See remarks.

`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 deserialized data

structure ([D3D12_ROOT_SIGNATURE_DESC1](#)). `ID3D12VersionedRootSignatureDeserializer` just owns the lifetime of the memory for the deserialized 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

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](#)

D3D12EnableExperimentalFeatures function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Enables a list of experimental features.

Syntax

```
HRESULT D3D12EnableExperimentalFeatures(
    UINT      NumFeatures,
    const IID *pIIDs,
    void     *pConfigurationStructs,
    UINT      *pConfigurationStructSizes
);
```

Parameters

NumFeatures

Type: **UINT**

The number of experimental features to enable.

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.

pConfigurationStructs

Type: **void***

SAL: `__in_ecount(NumFeatures)`

Structures that contain additional configuration details that some experimental features might need to be enabled.

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.

```
// -----
// -----
// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

Core Functions

D3D12GetDebugInterface function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a debug interface.

Syntax

```
HRESULT D3D12GetDebugInterface(  
    REFIID riid,  
    void    **ppvDebug  
>;
```

Parameters

`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.

`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.

```
// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

D3D12SerializeRootSignature function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Serializes a root signature version 1.0 that can be passed to [ID3D12Device::CreateRootSignature](#).

Syntax

```
HRESULT D3D12SerializeRootSignature(
    const D3D12_ROOT_SIGNATURE_DESC *pRootSignature,
    D3D_ROOT_SIGNATURE_VERSION      Version,
    ID3DBlob                      **ppBlob,
    ID3DBlob                      **ppErrorBlob
);
```

Parameters

pRootSignature

Type: [const D3D12_ROOT_SIGNATURE_DESC*](#)

The description of the root signature, as a pointer to a [D3D12_ROOT_SIGNATURE_DESC](#) structure.

Version

Type: [D3D_ROOT_SIGNATURE_VERSION](#)

A [D3D_ROOT_SIGNATURE_VERSION](#)-typed value that specifies the version of root signature.

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.

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 (when that capability is added), the compiled shader will contain a serialized root signature in it already.

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.

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Core Functions](#)

[Creating a Root Signature](#)

D3D12SerializeVersionedRootSignature function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Serializes a root signature of any version that can be passed to [ID3D12Device::CreateRootSignature](#).

Syntax

```
HRESULT D3D12SerializeVersionedRootSignature(
    const D3D12_VERSIONED_ROOT_SIGNATURE_DESC *pRootSignature,
    ID3DBlob                                **ppBlob,
    ID3DBlob                                **ppErrorBlob
);
```

Parameters

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.

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.

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 (when that capability is added), the compiled shader will contain a serialized root signature in it already.

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

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](#)

ID3D12CommandAllocator interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12CommandAllocator** interface has these methods.

METHOD	DESCRIPTION
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.

```
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;
```

Asynchronous compute thread.

```

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

Target Platform	Windows
Header	d3d12.h

See also

Core Interfaces

ID3D12Pageable

ID3D12CommandAllocator::Reset method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT Reset();
```

Parameters

This method has no parameters.

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

Apps call **Reset** to re-use the memory that is associated with a command allocator. From this call to **Reset**, the runtime and driver determine that the graphics processing unit (GPU) is no longer executing any command lists that have recorded commands with the command allocator.

Unlike [ID3D12GraphicsCommandList::Reset](#), it is not recommended that you call **Reset** on the command allocator while a command list is 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:

```
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;
```

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandAllocator](#)

ID3D12CommandList interface

5/13/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12CommandList** interface has these methods.

METHOD	DESCRIPTION
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:

```

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

Target Platform	Windows
Header	d3d12.h

See also

Core Interfaces

[ID3D12DeviceChild](#)

[ID3D12GraphicsCommandList](#)

Setting descriptor heaps

ID3D12CommandList::GetType method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_COMMAND_LIST_TYPE GetType();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandList](#)

ID3D12CommandQueue interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The ID3D12CommandQueue interface has these methods.

METHOD	DESCRIPTION
ID3D12CommandQueue::BeginEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command queue.
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::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::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.

```
// Compute objects.  
ComPtr<ID3D12CommandAllocator> m_computeAllocator[ThreadCount];  
ComPtr<ID3D12CommandQueue> m_computeCommandQueue[ThreadCount];  
ComPtr<ID3D12GraphicsCommandList> m_computeCommandList[ThreadCount];
```

Asynchronous compute thread.

```
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

ID3D12CommandQueue::BeginEvent method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void BeginEvent(  
    UINT      Metadata,  
    const void *pData,  
    UINT      Size  
) ;
```

Parameters

Metadata

Type: [UINT](#)

Internal.

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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::CopyTileMappings method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void CopyTileMappings(  
    ID3D12Resource                      *pDstResource,  
    const D3D12_TILED_RESOURCE_COORDINATE *pDstRegionStartCoordinate,  
    ID3D12Resource                      *pSrcResource,  
    const D3D12_TILED_RESOURCE_COORDINATE *pSrcRegionStartCoordinate,  
    const D3D12_TILE_REGION_SIZE          *pRegionSize,  
    D3D12_TILE_MAPPING_FLAGS             Flags  
) ;
```

Parameters

`pDstResource`

A pointer to the destination reserved resource.

`pDstRegionStartCoordinate`

A pointer to a [D3D12_TILED_RESOURCE_COORDINATE](#) structure that describes the starting coordinates of the destination reserved resource.

`pSrcResource`

A pointer to the source reserved resource.

`pSrcRegionStartCoordinate`

A pointer to a [D3D12_TILED_RESOURCE_COORDINATE](#) structure that describes the starting coordinates of the source reserved resource.

`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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

[UpdateTileMappings](#)

[Volume Tiled Resources](#)

ID3D12CommandQueue::EndEvent method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void EndEvent();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::ExecuteCommandLists method

7/1/2020 • 2 minutes to read • [Edit Online](#)

Submits an array of command lists for execution.

Syntax

```
void ExecuteCommandLists(
    UINT           NumCommandLists,
    ID3D12CommandList * const *ppCommandLists
);
```

Parameters

NumCommandLists

The number of command lists to be executed.

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](#) has not been called on any 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<IDXGISwapChain3> m_swapChain;  
ComPtr<ID3D11DeviceContext> m_d3d11DeviceContext;  
ComPtr<ID3D11On12Device> m_d3d11On12Device;  
ComPtr<ID3D12Device> m_d3d12Device;  
ComPtr<IDWriteFactory> m_dWriteFactory;  
ComPtr<ID2D1Factory3> m_d2dFactory;  
ComPtr<ID2D1Device2> m_d2dDevice;  
ComPtr<ID2D1DeviceContext2> 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;
```

```
// Render the scene.  
void D3D1211on12::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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::GetClockCalibration method

5/27/2020 • 2 minutes to read • [Edit Online](#)

This method samples the CPU and GPU timestamp counters at the same moment in time.

Syntax

```
HRESULT GetClockCalibration(  
    UINT64 *pGpuTimestamp,  
    UINT64 *pCpuTimestamp  
>;
```

Parameters

`pGpuTimestamp`

Type: `UINT64*`

The value of the GPU timestamp counter.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the description of the command queue.

Syntax

```
D3D12_COMMAND_QUEUE_DESC GetDesc();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_COMMAND_QUEUE_DESC](#)

The description of the command queue, as a [D3D12_COMMAND_QUEUE_DESC](#) structure.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::GetTimestampFrequency method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT GetTimestampFrequency(  
    UINT64 *pFrequency  
) ;
```

Parameters

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::SetMarker method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetMarker(  
    UINT      Metadata,  
    const void *pData,  
    UINT      Size  
) ;
```

Parameters

Metadata

Type: **UINT**

Internal.

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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12CommandQueue](#)

ID3D12CommandQueue::Signal method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Updates a fence to a specified value.

Syntax

```
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.

```
// 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);
```

```
// Add a signal command to the queue.  
ThrowIfFailed(m_commandQueue->Signal(m_renderContextFence.Get(), m_renderContextFenceValue));
```

```
// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandQueue](#)

[Multi-engine synchronization](#)

ID3D12CommandQueue::UpdateTileMappings method

5/27/2020 • 8 minutes to read • [Edit Online](#)

Updates mappings of tile locations in reserved resources to memory locations in a resource heap.

Syntax

```
void UpdateTileMappings(  
    ID3D12Resource                 *pResource,  
    UINT                           NumResourceRegions,  
    const D3D12_TILED_RESOURCE_COORDINATE *pResourceRegionStartCoordinates,  
    const D3D12_TILE_REGION_SIZE      *pResourceRegionSizes,  
    ID3D12Heap                      *pHeap,  
    UINT                           NumRanges,  
    const D3D12_TILE_RANGE_FLAGS     *pRangeFlags,  
    const UINT                       *pHeapRangeStartOffsets,  
    const UINT                       *pRangeTileCounts,  
    D3D12_TILE_MAPPING_FLAGS        Flags  
) ;
```

Parameters

pResource

A pointer to the reserved resource.

NumResourceRegions

The number of reserved resource regions.

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.

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.

pHeap

A pointer to the resource heap.

NumRanges

The number of tile ranges.

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.

pHeapRangeStartOffsets

An array of offsets into the resource heap. These are 0-based tile offsets, counting in tiles (not bytes).

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

```
// - NULL for pResourceRegionStartCoordinates and pResourceRegionSizes defaults to the entire resource
// - NULL for pHeapRangeStartOffsets 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:

```

// - 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,NULL,D3D12_TILE_MAPPING_FLAG_NONE);

```

Defining mappings for a set of disjoint individual tiles:

```

// - 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.

```

// - This complex example hard codes the parameter arrays, whereas in practice the
// application would likely configure the parameters programmatically 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 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

Target Platform	Windows
Header	d3d12.h

See also

[CopyTileMappings](#)

[ID3D12CommandQueue](#)

[Volume Tiled Resources](#)

ID3D12CommandQueue::Wait method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12CommandQueue](#)

[Multi-engine synchronization](#)

ID3D12CommandSignature interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

Methods

The ID3D12CommandSignature interface has these methods.

METHOD	DESCRIPTION

Remarks

To create a command signature, call [ID3D12Device::CreateCommandSignature](#), as described in [Indirect Drawing](#).

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ExecuteIndirect](#)

[ID3D12Pageable](#)

ID3D12DescriptorHeap interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#). **ID3D12DescriptorHeap** also has these types of members:

- [Methods](#)

Methods

The **ID3D12DescriptorHeap** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptor Heaps](#)

[Descriptor Heaps](#)

[ID3D12Pageable](#)

ID3D12DescriptorHeap::GetCPUDescriptorHandleForHeapStart method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_CPU_DESCRIPTOR_HANDLE GetCPUDescriptorHandleForHeapStart();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

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

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DescriptorHeap](#)

ID3D12DescriptorHeap::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the descriptor heap description.

Syntax

```
D3D12_DESCRIPTOR_HEAP_DESC GetDesc();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_DESCRIPTOR_HEAP_DESC](#)

The description of the descriptor heap, as a [D3D12_DESCRIPTOR_HEAP_DESC](#) structure.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12DescriptorHeap](#)

ID3D12DescriptorHeap::GetGPUDescriptorHandleForHeapStart method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_GPU_DESCRIPTOR_HANDLE GetGPUDescriptorHandleForHeapStart();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

Returns the GPU descriptor handle that represents the start of the heap.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DescriptorHeap](#)

ID3D12Device interface

6/30/2020 • 3 minutes to read • [Edit Online](#)

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 targeting Windows 10 should use this interface instead of later versions. Applications targeting 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](#)

Methods

The [ID3D12Device](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12Device::CheckFeatureSupport	Gets information about the features that are supported by the current graphics driver.
ID3D12Device::CopyDescriptors	Copies descriptors from a source to a destination.
ID3D12Device::CopyDescriptorsSimple	Copies descriptors from a source to a destination.
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.

METHOD	DESCRIPTION
ID3D12Device::CreateFence	Creates a fence object.
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::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::CreateSharedHandle	Creates a shared handle to an 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.

METHOD	DESCRIPTION
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.
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.

```
// 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.

```
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptors](#)

[ID3D12Device1](#)

[ID3D12Device2](#)

[ID3D12Object](#)

[Memory Management in Direct3D 12](#)

ID3D12Device::CheckFeatureSupport method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets information about the features that are supported by the current graphics driver.

Syntax

```
HRESULT CheckFeatureSupport(  
    D3D12_FEATURE Feature,  
    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.

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_D3D12_OPTIONS5](#) constant for the *Feature* parameter, and pass a pointer to a [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:

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CopyDescriptors method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Copies descriptors from a source to a destination.

Syntax

```
void CopyDescriptors(  
    UINT NumDestDescriptorRanges,  
    const D3D12_CPU_DESCRIPTOR_HANDLE *pDestDescriptorRangeStarts,  
    const UINT *pDestDescriptorRangeSizes,  
    UINT NumSrcDescriptorRanges,  
    const D3D12_CPU_DESCRIPTOR_HANDLE *pSrcDescriptorRangeStarts,  
    const UINT *pSrcDescriptorRangeSizes,  
    D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapsType  
) ;
```

Parameters

NumDestDescriptorRanges

Type: **UINT**

The number of destination descriptor ranges to copy to.

pDestDescriptorRangeStarts

Type: **const D3D12_CPU_DESCRIPTOR_HANDLE***

An array of CPU_descriptor_handle objects to copy to.

pDestDescriptorRangeSizes

Type: **const UINT***

An array of destination descriptor range sizes to copy to.

NumSrcDescriptorRanges

Type: **UINT**

The number of source descriptor ranges to copy from.

pSrcDescriptorRangeStarts

Type: **const D3D12_CPU_DESCRIPTOR_HANDLE***

An array of CPU_descriptor_handle objects to copy from.

pSrcDescriptorRangeSizes

Type: **const UINT***

An array of source descriptor range sizes to copy from.

DescriptorHeapsType

Type: [D3D12_DESCRIPTOR_HEAP_TYPE](#)

The [D3D12_DESCRIPTOR_HEAP_TYPE](#)-typed value that specifies the type of descriptor heap to copy with.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Copying Descriptors](#)

[ID3D12Device](#)

ID3D12Device::CopyDescriptorsSimple method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Copies descriptors from a source to a destination.

Syntax

```
void CopyDescriptorsSimple(  
    UINT                 NumDescriptors,  
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptorRangeStart,  
    D3D12_CPU_DESCRIPTOR_HANDLE SrcDescriptorRangeStart,  
    D3D12_DESCRIPTOR_HEAP_TYPE  DescriptorHeapsType  
) ;
```

Parameters

NumDescriptors

Type: [UINT](#)

The number of descriptors to copy.

DestDescriptorRangeStart

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A CPU_descriptor_handle that describes the destination descriptors to start to copy to.

SrcDescriptorRangeStart

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A CPU_descriptor_handle that describes the source descriptors to start to copy from.

DescriptorHeapsType

Type: [D3D12_DESCRIPTOR_HEAP_TYPE](#)

The [D3D12_DESCRIPTOR_HEAP_TYPE](#)-typed value that specifies the type of descriptor heap to copy with.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib

DLL	D3D12.dll
-----	-----------

See also

[Copying Descriptors](#)

[ID3D12Device](#)

ID3D12Device::CreateCommandAllocator method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a command allocator object.

Syntax

```
HRESULT CreateCommandAllocator(  
    D3D12_COMMAND_LIST_TYPE type,  
    REFIID                 riid,  
    void                  **ppCommandAllocator  
) ;
```

Parameters

`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.

`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:

```
ThrowIfFailed(pDevice->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,  
IID_PPV_ARGS(&m_commandAllocator)));  
ThrowIfFailed(pDevice->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_BUNDLE,  
IID_PPV_ARGS(&_bundleAllocator)));
```

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

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateCommandList method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a command list.

Syntax

```
HRESULT CreateCommandList(  
    UINT             nodeMask,  
    D3D12_COMMAND_LIST_TYPE type,  
    ID3D12CommandAllocator *pCommandAllocator,  
    ID3D12PipelineState   *pInitialState,  
    REFIID            riid,  
    void              **ppCommandList  
) ;
```

Parameters

`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](#).

`type`

Type: [D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of command list to create.

`pCommandAllocator`

Type: [ID3D12CommandAllocator*](#)

A pointer to the command allocator object from which the device creates command lists.

`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).

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the command list interface to return in *ppCommandList*.

```
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.

RETURN VALUE	DESCRIPTION
E_OUTOFMEMORY	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.

```
ComPtr<ID3D12CommandAllocator> m_commandAllocator;
ComPtr<ID3D12GraphicsCommandList> m_commandList;
```

Create a command allocator.

```
ThrowIfFailed(m_device->CreateCommandAllocator(D3D12_COMMAND_LIST_TYPE_DIRECT,
IID_PPV_ARGS(&m_commandAllocator)));
```

Creating the direct command list.

```
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

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[ID3D12GraphicsCommandList::Reset](#)

ID3D12Device::CreateCommandQueue method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a command queue.

Syntax

```
HRESULT CreateCommandQueue(  
    const D3D12_COMMAND_QUEUE_DESC *pDesc,  
    REFIID                   riid,  
    void                     **ppCommandQueue  
) ;
```

Parameters

pDesc

Type: **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: **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:

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateCommandSignature method

5/27/2020 • 2 minutes to read • [Edit Online](#)

This method creates a command signature.

Syntax

```
HRESULT CreateCommandSignature(  
    const D3D12_COMMAND_SIGNATURE_DESC *pDesc,  
    ID3D12RootSignature *pRootSignature,  
    REFIID riid,  
    void **ppvCommandSignature  
) ;
```

Parameters

pDesc

Type: **const D3D12_COMMAND_SIGNATURE_DESC***

Describes the command signature to be created with the **D3D12_COMMAND_SIGNATURE_DESC** structure.

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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateCommittedResource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT CreateCommittedResource(
    const D3D12_HEAP_PROPERTIES *pHeapProperties,
    D3D12_HEAP_FLAGS          HeapFlags,
    const D3D12_RESOURCE_DESC  *pDesc,
    D3D12_RESOURCE_STATES     InitialResourceState,
    const D3D12_CLEAR_VALUE   *pOptimizedClearValue,
    REFIID                   riidResource,
    void                     **ppvResource
);
```

Parameters

pHeapProperties

Type: [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_DESC*](#)

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

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`.

`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.

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.

Examples

The [D3D12Bundles](#) sample uses [ID3D12Device::CreateCommittedResource](#) as follows:

Create a vertex buffer.

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[CreatePlacedResource](#)

[CreateReservedResource](#)

[D3D12_HEAP_FLAGS](#)

[ID3D12Device](#)

ID3D12Device::CreateComputePipelineState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a compute pipeline state object.

Syntax

```
HRESULT CreateComputePipelineState(
    const D3D12_COMPUTE_PIPELINE_STATE_DESC *pDesc,
    REFIID                           riid,
    void                            **ppPipelineState
);
```

Parameters

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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib

DLL	D3D12.dll
-----	-----------

See also

[ID3D12Device](#)

ID3D12Device::CreateConstantBufferView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a constant-buffer view for accessing resource data.

Syntax

```
void CreateConstantBufferView(
    const D3D12_CONSTANT_BUFFER_VIEW_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor
);
```

Parameters

pDesc

Type: [const D3D12_CONSTANT_BUFFER_VIEW_DESC*](#)

A pointer to a [D3D12_CONSTANT_BUFFER_VIEW_DESC](#) structure that describes the constant-buffer view.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateDepthStencilView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a depth-stencil view for accessing resource data.

Syntax

```
void CreateDepthStencilView(  
    ID3D12Resource                 *pResource,  
    const D3D12_DEPTH_STENCIL_VIEW_DESC *pDesc,  
    D3D12_CPU_DESCRIPTOR_HANDLE      DestDescriptor  
) ;
```

Parameters

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.

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.

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

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateDescriptorHeap method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a descriptor heap object.

Syntax

```
HRESULT CreateDescriptorHeap(  
    const D3D12_DESCRIPTOR_HEAP_DESC *pDescriptorHeapDesc,  
    REFIID  
        iid,  
    void  
        **ppvHeap  
) ;
```

Parameters

`pDescriptorHeapDesc`

Type: `const D3D12_DESCRIPTOR_HEAP_DESC*`

A pointer to a `D3D12_DESCRIPTOR_HEAP_DESC` structure that describes the heap.

`iid`

Type: `REFIID`

The globally unique identifier (**GUID**) for the descriptor heap interface. See Remarks. An input parameter.

`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.

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateFence method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a fence object.

Syntax

```
HRESULT CreateFence(
    UINT64           initialValue,
    D3D12_FENCE_FLAGS Flags,
    REFIID           riid,
    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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateGraphicsPipelineState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a graphics pipeline state object.

Syntax

```
HRESULT CreateGraphicsPipelineState(  
    const D3D12_GRAPHICS_PIPELINE_STATE_DESC *pDesc,  
    REFIID  
                    iid,  
    void  
        **ppPipelineState  
) ;
```

Parameters

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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib

DLL	D3D12.dll
-----	-----------

See also

[ID3D12Device](#)

ID3D12Device::CreateHeap method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a heap that can be used with placed resources and reserved resources.

Syntax

```
HRESULT CreateHeap(  
    const D3D12_HEAP_DESC *pDesc,  
    REFIID                 riid,  
    void                  **ppvHeap  
) ;
```

Parameters

pDesc

Type: **const D3D12_HEAP_DESC***

A pointer to a constant **D3D12_HEAP_DESC** structure that describes the heap.

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**.

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.

RETURN VALUE	DESCRIPTION
E_OUTOFMEMORY	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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[Shared heaps](#)

ID3D12Device::CreatePlacedResource method

5/13/2020 • 6 minutes to read • [Edit Online](#)

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

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[CreateCommittedResource](#)

[CreateReservedResource](#)

[ID3D12Device](#)

[Shared Heaps](#)

ID3D12Device::CreateQueryHeap method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT CreateQueryHeap(  
    const D3D12_QUERY_HEAP_DESC *pDesc,  
    REFIID                   riid,  
    void                     **ppvHeap  
) ;
```

Parameters

pDesc

Type: **const D3D12_QUERY_HEAP_DESC***

Specifies the query heap in a **D3D12_QUERY_HEAP_DESC** structure.

riid

Type: **REFIID**

Specifies a REFIID that uniquely identifies the heap.

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.

```

// 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;

```

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateRenderTargetView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a render-target view for accessing resource data.

Syntax

```
void CreateRenderTargetView(  
    ID3D12Resource                 *pResource,  
    const D3D12_RENDER_TARGET_VIEW_DESC *pDesc,  
    D3D12_CPU_DESCRIPTOR_HANDLE      DestDescriptor  
) ;
```

Parameters

`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.

`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.

`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

Target Platform	Windows

Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateReservedResource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a resource that is reserved, and not yet mapped to any pages in a heap.

Syntax

```
HRESULT CreateReservedResource(
    const D3D12_RESOURCE_DESC *pDesc,
    D3D12_RESOURCE_STATES     InitialState,
    const D3D12_CLEAR_VALUE   *pOptimizedClearValue,
    REFIID                   riid,
    void                     **ppvResource
);
```

Parameters

`pDesc`

Type: `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.

`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`.

`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](#).

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

Target Platform	Windows
Header	<code>d3d12.h</code>
Library	<code>D3D12.lib</code>
DLL	<code>D3D12.dll</code>

See also

[CreateCommittedResource](#)

[CreatePlacedResource](#)

[ID3D12Device](#)

ID3D12Device::CreateRootSignature method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a root signature layout.

Syntax

```
HRESULT CreateRootSignature(  
    UINT          nodeMask,  
    const void * pBlobWithRootSignature,  
    SIZE_T        blobLengthInBytes,  
    REFIID        riid,  
    void         **ppvRootSignature  
) ;
```

Parameters

`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](#).

`pBlobWithRootSignature`

Type: [const void*](#)

A pointer to the source data for the serialized signature.

`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.

`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.

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateSampler method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Create a sampler object that encapsulates sampling information for a texture.

Syntax

```
void CreateSampler(  
    const D3D12_SAMPLER_DESC      *pDesc,  
    D3D12_CPU_DESCRIPTOR_HANDLE DestDescriptor  
)
```

Parameters

pDesc

Type: [const D3D12_SAMPLER_DESC*](#)

A pointer to a [D3D12_SAMPLER_DESC](#) structure that describes the sampler.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateShaderResourceView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a shader-resource view for accessing data in a resource.

Syntax

```
void CreateShaderResourceView(
    ID3D12Resource                 *pResource,
    const D3D12_SHADER_RESOURCE_VIEW_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE       DestDescriptor
);
```

Parameters

`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.

`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.

`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.

```
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_srvUavHeap->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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateSharedHandle method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a shared handle to an heap, resource, or fence object.

Syntax

```
HRESULT CreateSharedHandle(  
    ID3D12DeviceChild          *pObject,  
    const SECURITY_ATTRIBUTES   *pAttributes,  
    DWORD                      Access,  
    LPCWSTR                     Name,  
    HANDLE                     *pHandle  
) ;
```

Parameters

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](#)

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 **lpSecurityDescriptor** member of the structure specifies a [SECURITY_DESCRIPTOR](#) for the resource. Set this member to **NULL** if you want the runtime to assign a default security descriptor to the resource that is associated with the returned handle. The ACLs in the default security descriptor for the resource come from the primary or impersonation token of the creator. For more info, see [Synchronization Object Security and Access Rights](#).

Access

Type: [DWORD](#)

Currently the only value this parameter accepts is [GENERIC_ALL](#).

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](#).

`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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::CreateUnorderedAccessView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a view for unordered accessing.

Syntax

```
void CreateUnorderedAccessView(
    ID3D12Resource                 *pResource,
    ID3D12Resource                 *pCounterResource,
    const D3D12_UNORDERED_ACCESS_VIEW_DESC *pDesc,
    D3D12_CPU_DESCRIPTOR_HANDLE     DestDescriptor
);
```

Parameters

pResource

Type: [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 D3D11-like null binding behavior (reading 0s, writes are discarded), but must have a valid *pDesc* in order to determine the descriptor type.

pCounterResource

Type: [ID3D12Resource](#)*

The [ID3D12Resource](#) for the counter (if any) associated with the UAV.

If *pCounterResource* is not specified, 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 4 bytes, and must be within the range of the counter resource.
- *pResource* cannot be NULL
- *pDesc* cannot be NULL.

pDesc

Type: [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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::Evict method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Enables the page-out of data, which precludes GPU access of that data.

Syntax

```
HRESULT Evict(  
    UINT           NumObjects,  
    ID3D12Pageable * const *ppObjects  
) ;
```

Parameters

NumObjects

Type: [UINT](#)

The number of objects in the *ppObjects* array to evict from the device.

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

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetAdapterLuid method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a locally unique identifier for the current device (adapter).

Syntax

```
LUID GetAdapterLuid();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetCopyableFootprints method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void GetCopyableFootprints(
    const D3D12_RESOURCE_DESC           *pResourceDesc,
    UINT                                FirstSubresource,
    UINT                                NumSubresources,
    UINT64                             BaseOffset,
    D3D12_PLACED_SUBRESOURCE_FOOTPRINT *pLayouts,
    UINT                                *pNumRows,
    UINT64                             *pRowSizeInBytes,
    UINT64                             *pTotalBytes
);
```

Parameters

pResourceDesc

Type: **const D3D12_RESOURCE_DESC***

A description of the resource, as a pointer to a [D3D12_RESOURCE_DESC](#) structure.

FirstSubresource

Type: **UINT**

Index of the first subresource in the resource. The range of valid values is 0 to [D3D12_REQ_SUBRESOURCES](#).

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.

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.

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.

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.

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:

```
// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CD3DX12_RESOURCE_DESC](#)

[CD3DX12_SUBRESOURCE_FOOTPRINT](#)

[ID3D12Device](#)

ID3D12Device::GetCustomHeapProperties method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_HEAP_PROPERTIES GetCustomHeapProperties(  
    UINT           nodeMask,  
    D3D12_HEAP_TYPE heapType  
)
```

Parameters

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:

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:

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:

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetDescriptorHandleIncrementSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
UINT GetDescriptorHandleIncrementSize(  
    D3D12_DESCRIPTOR_HEAP_TYPE DescriptorHeapType  
)
```

Parameters

`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.

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib

DLL	D3D12.dll
-----	-----------

See also

[ID3D12Device](#)

ID3D12Device::GetDeviceRemovedReason method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the reason that the device was removed.

Syntax

```
HRESULT GetDeviceRemovedReason();
```

Parameters

This method has no parameters.

Return value

Type: [HRESULT](#)

This method returns the reason that the device was removed.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetNodeCount method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT GetNodeCount();
```

Parameters

This method has no parameters.

Return value

Type: [UINT](#)

The number of physical adapters (nodes) that this device has.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetResourceAllocationInfo method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the size and alignment of memory required for a collection of resources on this adapter.

Syntax

```
D3D12_RESOURCE_ALLOCATION_INFO GetResourceAllocationInfo(  
    UINT           visibleMask,  
    UINT           numResourceDescs,  
    const D3D12_RESOURCE_DESC *pResourceDescs  
) ;
```

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.

`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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::GetResourceTiling method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets info about how a tiled resource is broken into tiles.

Syntax

```
void GetResourceTiling(
    ID3D12Resource           *pTiledResource,
    UINT                      *pNumTilesForEntireResource,
    D3D12_PACKED_MIP_INFO    *pPackedMipDesc,
    D3D12_TILE_SHAPE          *pStandardTileShapeForNonPackedMips,
    UINT                      *pNumSubresourceTilings,
    UINT                      FirstSubresourceTilingToGet,
    D3D12_SUBRESOURCE_TILING *pSubresourceTilingsForNonPackedMips
);
```

Parameters

`pTiledResource`

Type: [ID3D12Resource*](#)

Specifies a tiled [ID3D12Resource](#) to get info about.

`pNumTilesForEntireResource`

Type: `UINT*`

A pointer to a variable that receives the number of tiles needed to store the entire tiled resource.

`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.

`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.

`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).

`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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device](#)

[Subresources](#)

ID3D12Device::MakeResident method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Makes objects resident for the device.

Syntax

```
HRESULT MakeResident(  
    UINT           NumObjects,  
    ID3D12Pageable * const *ppObjects  
) ;
```

Parameters

`NumObjects`

Type: [UINT](#)

The number of objects in the *ppObjects* array to make resident for the device.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device::OpenSharedHandle method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT OpenSharedHandle(  
    HANDLE NTHandle,  
    REFIID riid,  
    void    **ppvObj  
) ;
```

Parameters

`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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[Multi-adapter systems](#)

ID3D12Device::OpenSharedHandleByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT OpenSharedHandleByName(  
    LPCWSTR Name,  
    DWORD   Access,  
    HANDLE  *pNTHandle  
) ;
```

Parameters

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](#).

pNTHandle

Type: **HANDLE***

Pointer to the shared handle.

Return value

Type: **HRESULT**

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

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

ID3D12Device

ID3D12Device::SetStablePowerState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

A development-time aid for certain types of profiling and experimental prototyping.

Syntax

```
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

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

ID3D12Device1 interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The [ID3D12Device1](#) interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

[ID3D12Device2](#)

ID3D12Device1::CreatePipelineLibrary method

6/25/2020 • 2 minutes to read • [Edit Online](#)

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 from being written to disk.

Syntax

```
HRESULT CreatePipelineLibrary(
    const void *pLibraryBlob,
    SIZE_T     BlobLength,
    REFIID     riid,
    void       **ppPipelineLibrary
);
```

Parameters

`pLibraryBlob`

Type: `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: `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

Create a PSO library and add PSOs to it. Note the macro `IID_PPV_ARGS` expands to become two parameters.

```
ID3D12Device* Device;
VERIFY_SUCCEEDED(D3D12CreateDevice(nullptr, IID_PPV_ARGS(&Device)));
ID3D12PipelineState* PS01, PS02;

// Fill out the PSO descs and then call CreateGraphicsPipelineState or CreateComputePipelineState

ID3D12PipelineLibrary* Library;
VERIFY_SUCCEEDED(Device->CreatePipelineLibrary(nullptr, 0, IID_PPV_ARGS(&Library)));
VERIFY_SUCCEEDED(Library->StorePipeline(L"PS01", PS01));
VERIFY_SUCCEEDED(Library->StorePipeline(L"PS02", PS02));
SIZE_T LibrarySize = Library->GetSerializedSize();
void* pData = new BYTE[LibrarySize];
VERIFY_SUCCEEDED(Library->Serialize(pData, LibrarySize));

// Save pData to disk
...
```

Create a PSO library using data loaded off of disk and retrieve PSOs out of it. This time the call to [CreatePipelineLibrary](#) de-serializes the library.

```

ID3D12Device* Device;
VERIFY_SUCCEEDED(D3D12CreateDevice(nullptr, IID_PPV_ARGS(&Device)));
ID3D12PipelineState* PS01, PS02;
const void* LibraryData;
SIZE_T LibraryDataSize;

// Load library data from disk

ID3D12PipelineLibrary* Library;
VERIFY_SUCCEEDED(Device->CreatePipelineLibrary(LibraryData, LibraryDataSize, IID_PPV_ARGS(&Library)));
VERIFY_SUCCEEDED(Library->LoadGraphicsPipeline(L“PS01”, IID_PPV_ARGS(&PS01)));
VERIFY_SUCCEEDED(Library->LoadComputePipeline(L“PS02”, IID_PPV_ARGS(&PS02)));

```

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device1, Pipeline State Cache sample](#)

ID3D12Device1::SetEventOnMultipleFenceCompletion method

5/5/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT SetEventOnMultipleFenceCompletion(
    ID3D12Fence* ppFences,
    const UINT64* pFenceValues,
    UINT NumFences,
    D3D12_MULTIPLE_FENCE_WAIT_FLAGS Flags,
    HANDLE hEvent
);
```

Parameters

ppFences

Type: **ID3D12Fence***

An array of length *NumFences* that specifies the **ID3D12Fence** objects.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device1](#)

ID3D12Device1::SetResidencyPriority method

5/27/2020 • 2 minutes to read • [Edit Online](#)

This method sets residency priorities of a specified list of objects.

Syntax

```
HRESULT SetResidencyPriority(  
    UINT                 NumObjects,  
    ID3D12Pageable*     * ppObjects,  
    const D3D12_RESIDENCY_PRIORITY* pPriorities  
) ;
```

Parameters

NumObjects

Type: **UINT**

Specifies the number of objects in the *ppObjects* and *pPriorities* arrays.

ppObjects

Type: **ID3D12Pageable***

Specifies an array, of length *NumObjects*, containing references to **ID3D12Pageable** objects.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

DLL	D3d12.dll
-----	-----------

See also

[ID3D12Device1](#)

ID3D12Device2 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The [ID3D12Device2](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12Device2::CreatePipelineState	Creates a pipeline state object from a pipeline state stream description.

Remarks

Use [D3D12CreateDevice](#) to create a device.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

[ID3D12Device1](#)

ID3D12Device2::CreatePipelineState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT CreatePipelineState(  
    const D3D12_PIPELINE_STATE_STREAM_DESC *pDesc,  
    REFIID                      riid,  
    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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device2](#)

ID3D12Device3 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The [ID3D12Device3](#) interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Device](#)

ID3D12Device1

ID3D12Device2

ID3D12Device3::EnqueueMakeResident method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Asynchronously makes objects resident for the device.

Syntax

```
HRESULT EnqueueMakeResident(
    D3D12_RESIDENCY_FLAGS Flags,
    UINT NumObjects,
    ID3D12Pageable * const *ppObjects,
    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.

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

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Device](#)

[ID3D12Device3](#)

ID3D12Device3::OpenExistingHeapFromAddress method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT OpenExistingHeapFromAddress(
    const void *pAddress,
    REFIID     riid,
    void       **ppvHeap
);
```

Parameters

`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.

`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.

Heaps 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device3 interface](#)

ID3D12Device3::OpenExistingHeapFromFileMapping method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT OpenExistingHeapFromFileMapping(  
    HANDLE hFileMapping,  
    REFIID riid,  
    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.

`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.

Heaps 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Device3 interface](#)

ID3D12Device4 interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

Represents a virtual adapter.

This interface extends [ID3D12Device3](#).

Inheritance

The ID3D12Device4 interface inherits from the ID3D12Device3 interface.

Methods

The **ID3D12Device4** interface has these methods.

METHOD	DESCRIPTION
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::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.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

ID3D12Device4::CreateCommandList1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a command list in the closed state. Also see [ID3D12Device::CreateCommandList](#).

Syntax

```
HRESULT CreateCommandList1(
    UINT             nodeMask,
    D3D12_COMMAND_LIST_TYPE type,
    D3D12_COMMAND_LIST_FLAGS flags,
    REFIID          riid,
    void            **ppCommandList
);
```

Parameters

`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](#).

`type`

Type: [D3D12_COMMAND_LIST_TYPE](#)

Specifies the type of command list to create.

`flags`

Type: [D3D12_COMMAND_LIST_FLAGS](#)

Specifies creation flags.

`riid`

Type: [REFIID](#)

A reference to the globally unique identifier (GUID) of the command list interface to return in *ppCommandList*.

`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.

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

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[ID3D12Device::CreateCommandList](#)

ID3D12Device4::CreateCommittedResource1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT CreateCommittedResource1(
    const D3D12_HEAP_PROPERTIES      *pHeapProperties,
    D3D12_HEAP_FLAGS                HeapFlags,
    const D3D12_RESOURCE_DESC        *pDesc,
    D3D12_RESOURCE_STATES           InitialResourceState,
    const D3D12_CLEAR_VALUE          *pOptimizedClearValue,
    ID3D12ProtectedResourceSession *pProtectedSession,
    REFIID                          riidResource,
    void                            **ppvResource
);
```

Parameters

`pHeapProperties`

Type: [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_DESC*](#)

A pointer to a [D3D12_RESOURCE_DESC](#) structure that describes the resource.

`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.

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.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

ID3D12Device4::CreateHeap1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a heap (optionally for a protected session) that can be used with placed resources and reserved resources.
Also see [ID3D12Device::CreateHeap](#).

Syntax

```
HRESULT CreateHeap1(
    const D3D12_HEAP_DESC           *pDesc,
    ID3D12ProtectedResourceSession *pProtectedSession,
    REFIID                          riid,
    void                            **ppvHeap
);
```

Parameters

pDesc

Type: [const D3D12_HEAP_DESC*](#)

A pointer to a constant **D3D12_HEAP_DESC** structure that describes the heap.

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](#).

riid

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the heap interface to return in *ppvHeap*.

While *riid* 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](#).

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.

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

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

ID3D12Device4::CreateProtectedResourceSession method

6/25/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT CreateProtectedResourceSession(  
    const D3D12_PROTECTED_RESOURCE_SESSION_DESC *pDesc,  
    REFIID  
                riid,  
    void  
                **ppSession  
) ;
```

Parameters

`pDesc`

Type: `const D3D12_PROTECTED_RESOURCE_SESSION_DESC*`

A pointer to a constant `D3D12_PROTECTED_RESOURCE_SESSION_DESC` structure, describing the session to create.

`riid`

Type: `REFIID`

A reference to the globally unique identifier (GUID) of the `ID3D12ProtectedResourceSession` interface.

`ppSession`

Type: `void**`

A pointer to a memory block that receives an `ID3D12ProtectedResourceSession` interface pointer to the created session object.

Return value

None

Requirements

Target Platform	Windows
-----------------	---------

Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

ID3D12Device4::CreateReservedResource1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT CreateReservedResource1(
    const D3D12_RESOURCE_DESC      *pDesc,
    D3D12_RESOURCE_STATES        InitialState,
    const D3D12_CLEAR_VALUE       *pOptimizedClearValue,
    ID3D12ProtectedResourceSession *pProtectedSession,
    REFIID                         riid,
    void                           **ppvResource
);
```

Parameters

pDesc

Type: [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.

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](#).

`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.

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

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[CreateCommittedResource1](#)

[CreatePlacedResource](#)

[ID3D12Device4](#)

ID3D12Device4::GetResourceAllocationInfo1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
D3D12_RESOURCE_ALLOCATION_INFO GetResourceAllocationInfo1(
    UINT                      visibleMask,
    UINT                      numResourceDescs,
    const D3D12_RESOURCE_DESC *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_DESC*](#)

An array of [D3D12_RESOURCE_DESC](#) 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

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

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

See also

[ID3D12Device4](#)

ID3D12Device5 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12Device5::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.
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

Minimum supported client	Windows 10, version 1809
Minimum supported server	Windows Server 2016
Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

ID3D12Device5::CheckDriverMatchingIdentifier method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
D3D12_DRIVER_MATCHING_IDENTIFIER_STATUS CheckDriverMatchingIdentifier(  
    D3D12_SERIALIZED_DATA_TYPE SerializedDataType,  
    const D3D12_SERIALIZED_DATA_DRIVER_MATCHING_IDENTIFIER *pIdentifierToCheck  
) ;
```

Parameters

SerializedDataType

The type of the serialized data. For more information, see [D3D12_SERIALIZED_DATA_TYPE](#).

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

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](#)

ID3D12Device5::CreateLifetimeTracker method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT CreateLifetimeTracker(  
    ID3D12LifetimeOwner *pOwner,  
    REFIID             riid,  
    void              **ppvTracker  
) ;
```

Parameters

pOwner

Type: **ID3D12LifetimeOwner***

A pointer to an **ID3D12LifetimeOwner** interface representing the application-defined callback.

riid

Type: **REFIID**

A reference to the interface identifier (IID) of the interface to return in *ppvTracker*.

ppvTracker

Type: **void****

A pointer to a memory block that receives the requested interface pointer to the created object.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12Device5::CreateMetaCommand method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Creates an instance of the specified meta command.

Syntax

```
HRESULT CreateMetaCommand(  
    REFGUID    CommandId,  
    UINT       NodeMask,  
    const void *pCreationParametersData,  
    SIZE_T     CreationParametersDataSizeInBytes,  
    REFIID     riid,  
    void       **ppMetaCommand  
) ;
```

Parameters

`CommandId`

Type: **REFIID**

A reference to the globally unique identifier (GUID) of the meta command that you wish to instantiate.

`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](#).

`pCreationParametersData`

Type: **const void***

An optional pointer to a constant structure containing the values of the parameters for creating the meta command.

`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](#).

`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.

RETURN VALUE	DESCRIPTION
DXG I_ER ROR _UN SUP POR TED	The current hardware does not support the algorithm being requested

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

ID3D12Device5::CreateStateObject method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates an [ID3D12StateObject](#).

Syntax

```
HRESULT CreateStateObject(  
    const D3D12_STATE_OBJECT_DESC *pDesc,  
    REFIID                   riid,  
    void                     **ppStateObject  
) ;
```

Parameters

`pDesc`

The description of the state object to create.

`riid`

The GUID of the interface to create. Use `_uuidof(ID3D12StateObject)`.

`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

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](#)

ID3D12Device5::EnumerateMetaCommandParameters method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Queries reflection metadata about the parameters of the specified meta command.

Syntax

```
HRESULT EnumerateMetaCommandParameters(
    REFGUID CommandId,
    D3D12_META_COMMAND_PARAMETER_STAGE Stage,
    UINT *pTotalStructureSizeInBytes,
    UINT *pParameterCount,
    D3D12_META_COMMAND_PARAMETER_DESC *pParameterDescs
);
```

Parameters

`CommandId`

Type: `REFIID`

A reference to the globally unique identifier (GUID) of the meta command whose parameters you wish to be returned in *pParameterDescs*.

`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.

`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.

`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`.

`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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

ID3D12Device5::EnumerateMetaCommands method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Queries reflection metadata about available meta commands.

Syntax

```
HRESULT EnumerateMetaCommands(
    UINT                 *pNumMetaCommands,
    D3D12_META_COMMAND_DESC *pDescs
);
```

Parameters

`pNumMetaCommands`

Type: `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: `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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Device5](#)

ID3D12Device5::GetRaytracingAccelerationStructurePrebuildInfo method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void GetRaytracingAccelerationStructurePrebuildInfo(
    const D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INPUTS *pDesc,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO      *pInfo
);
```

Parameters

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.

pInfo

The result of the query.

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

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](#)

ID3D12Device5::RemoveDevice method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void RemoveDevice();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12Device6 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a virtual adapter.

This interface extends [ID3D12Device5](#).

Inheritance

The ID3D12Device6 interface inherits from the ID3D12Device5 interface.

Methods

The **ID3D12Device6** interface has these methods.

METHOD	DESCRIPTION
ID3D12Device6::SetBackgroundProcessingMode	Sets the mode for driver background processing optimizations.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core interfaces](#)

ID3D12Device6::SetBackgroundProcessingMode method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the mode for driver background processing optimizations.

Syntax

```
HRESULT SetBackgroundProcessingMode(  
    D3D12_BACKGROUND_PROCESSING_MODE Mode,  
    D3D12_MEASUREMENTS_ACTION MeasurementsAction,  
    HANDLE hEventToSignalUponCompletion,  
    BOOL *pbFurtherMeasurementsDesired  
>;
```

Parameters

Mode

Type: [D3D12_BACKGROUND_PROCESSING_MODE](#)

The level of dynamic optimization to apply to GPU work that's subsequently submitted.

MeasurementsAction

Type: [D3D12_MEASUREMENTS_ACTION](#)

The action to take with the results of earlier workload instrumentation.

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.

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`.

Return value

None

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.

```
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

Target Platform	Windows
Header	d3d12.h
Library	d3d12.lib
DLL	d3d12.dll

ID3D12DeviceChild interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12DeviceChild** interface has these methods.

METHOD	DESCRIPTION
ID3D12DeviceChild::GetDevice	Gets a pointer to the device that created this interface.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Object](#)

ID3D12DeviceChild::GetDevice method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a pointer to the device that created this interface.

Syntax

```
HRESULT GetDevice(  
    REFIID riid,  
    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.

`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:

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12DeviceChild](#)

ID3D12DeviceRemovedExtendedData interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

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

ID3D12DeviceRemovedExtendedData::GetAutoBreadcrumbsOutput method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the Device Removed Extended Data (DRED) auto-breadcrumbs output after device removal.

Syntax

```
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

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](#)

ID3D12DeviceRemovedExtendedData::GetPageFaultAllocationOutput method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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](#)

ID3D12DeviceRemovedExtendedDataSettings interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

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

ID3D12DeviceRemovedExtendedDataSettings::SetAutoBreadcrumbsEnablement method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

ID3D12DeviceRemovedExtendedDataSettings::SetPageFaultEnablement method

5/13/2020 • 2 minutes to read • [Edit Online](#)

Configures the enablement settings for Device Removed Extended Data (DRED) page fault reporting.

Syntax

```
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

Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

ID3D12DeviceRemovedExtendedDataSettings::SetWatsonDumpEnablement method

5/13/2020 • 2 minutes to read • [Edit Online](#)

Configures the enablement settings for Device Removed Extended Data (DRED) dump creation for [Windows Error Reporting \(WER\)](#), also known as Watson.

Syntax

```
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

Minimum supported client	Windows 10, version 1903
Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

- [ID3D12DeviceRemovedExtendedDataSettings interface](#)
- [Use DRED to diagnose GPU faults](#)

ID3D12Fence interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12Fence** interface has these methods.

METHOD	DESCRIPTION
ID3D12Fence::GetCompletedValue	Gets the current value of the fence.
ID3D12Fence::SetEventOnCompletion	Specifies an event that should be fired when the fence reaches a certain value.
ID3D12Fence::Signal	Sets the fence to the specified value.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Fence Based Resource Management](#)

[ID3D12Pageable](#)

[Multi-engine synchronization](#)

ID3D12Fence::GetCompletedValue method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the current value of the fence.

Syntax

```
UINT64 GetCompletedValue();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Fence](#)

[Multi-engine synchronization](#)

ID3D12Fence::SetEventOnCompletion method

5/5/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

Examples

The [D3D12Multithreading](#) sample uses **ID3D12Fence::SetEventOnCompletion** as follows:

```

// 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 D3D12 Reference](#).

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Fence](#)

[Multi-engine synchronization](#)

ID3D12Fence::Signal method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the fence to the specified value.

Syntax

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Fence](#)

[Multi-engine synchronization](#)

ID3D12Fence1 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The `ID3D12Fence1` interface has these methods.

METHOD	DESCRIPTION
ID3D12Fence1::GetCreationFlags	Gets the flags used to create the fence represented by the current instance.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Fence Based Resource Management](#)

[ID3D12fence](#)

[Multi-engine synchronization](#)

ID3D12Fence1::GetCreationFlags method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_FENCE_FLAGS GetCreationFlags();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Id3d12fence1](#)

[Multi-engine synchronization](#)

ID3D12GraphicsCommandList interface

6/30/2020 • 5 minutes to read • [Edit Online](#)

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 targeting 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](#)

Methods

The [ID3D12GraphicsCommandList](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12GraphicsCommandList::BeginEvent	Not intended to be called directly. Use the PIX event runtime to insert events into a command list.
ID3D12GraphicsCommandList::BeginQuery	Starts a query running.
ID3D12GraphicsCommandList::ClearDepthStencilView	Clears the depth-stencil resource.
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::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::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.

METHOD	DESCRIPTION
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::DiscardResource	Discards a resource.
ID3D12GraphicsCommandList::Dispatch	Executes a command list from 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::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::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::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.

METHOD	DESCRIPTION
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::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::SetPipelineState	Sets all shaders and programs most of the fixed-function state of the graphics processing unit (GPU) pipeline.

METHOD	DESCRIPTION
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.

```
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.

```
// 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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12CommandList](#)

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList::BeginEvent method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void BeginEvent(  
    UINT      Metadata,  
    const void *pData,  
    UINT      Size  
) ;
```

Parameters

Metadata

Type: [UINT](#)

Internal.

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

Target Platform	Windows
Header	d3d12.h

Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::BeginQuery method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Starts a query running.

Syntax

```
void BeginQuery(  
    ID3D12QueryHeap  *pQueryHeap,  
    D3D12_QUERY_TYPE Type,  
    UINT              Index  
) ;
```

Parameters

pQueryHeap

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the query.

Type

Type: [D3D12_QUERY_TYPE](#)

Specifies one member of [D3D12_QUERY_TYPE](#).

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:

```
// 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()));
```

```

    m_commandList->Reset(m_commandAllocator, 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ClearDepthStencilView method

5/27/2020 • 3 minutes to read • [Edit Online](#)

Clears the depth-stencil resource.

Syntax

```
void ClearDepthStencilView(  
    D3D12_CPU_DESCRIPTOR_HANDLE DepthStencilView,  
    D3D12_CLEAR_FLAGS           ClearFlags,  
    FLOAT                      Depth,  
    UINT8                      Stencil,  
    UINT                       NumRects,  
    const D3D12_RECT*          *pRects  
)
```

Parameters

`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.

`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).

`Depth`

Type: [FLOAT](#)

A value to clear the depth buffer with. This value will be clamped between 0 and 1.

`Stencil`

Type: [UINT8](#)

A value to clear the stencil buffer with.

`NumRects`

Type: [UINT](#)

The number of rectangles in the array that the `pRects` parameter specifies.

`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

[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:

```
// 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;
```

```

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:

```

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()));
    }
}

```

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ClearRenderTargetView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets all the elements in a render target to one value.

Syntax

```
void ClearRenderTargetView(  
    D3D12_CPU_DESCRIPTOR_HANDLE RenderTargetView,  
    const FLOAT [4]           ColorRGBA,  
    UINT                     NumRects,  
    const D3D12_RECT*         *pRects  
) ;
```

Parameters

`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.

`ColorRGBA`

Type: `const FLOAT[4]`

A 4-component array that represents the color to fill the render target with.

`NumRects`

Type: `UINT`

The number of rectangles in the array that the `pRects` parameter specifies.

`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:

```
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;
```

```

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:

```

// Frame resources.
FrameResource* m_frameResources[FrameCount];
FrameResource* m_pCurrentFrameResource;
int m_currentFrameResourceIndex;

```

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ClearState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Resets the state of a direct command list back to the state it was in when the command list was created.

Syntax

```
void ClearState(  
    ID3D12PipelineState *pPipelineState  
>);
```

Parameters

`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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ClearUnorderedAccessViewFloat method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets all the elements in a unordered access view to the specified float values.

Syntax

```
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: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

Describes the GPU descriptor handle that represents the start of the heap for the unordered-access view to clear.

`ViewCPUHandle`

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

Describes the CPU descriptor handle that represents the start of the heap for the render target to clear.

`pResource`

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the unordered-access-view resource to clear.

`Values`

Type: [const FLOAT\[4\]](#)

A 4-component array that containing the values to fill the unordered-access-view resource with.

`NumRects`

Type: [UINT](#)

The number of rectangles in the array that the `pRects` parameter specifies.

`pRects`

Type: [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 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 values are outside of a normalized range.

The debug layer will issue an error if the subresources referenced by the view are not in the appropriate state. For [ClearUnorderedAccessViewFloat](#), the state must be [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#).

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ClearUnorderedAccessViewUint method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets all the elements in a unordered-access view (UAV) to the specified integer values.

Syntax

```
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: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A [D3D12_GPU_DESCRIPTOR_HANDLE](#) that references an initialized descriptor for the unordered-access view 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](#).

`ViewCPUHandle`

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A [D3D12_CPU_DESCRIPTOR_HANDLE](#) 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.

`pResource`

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the unordered-access view resource to clear.

`Values`

Type: [const uint\[4\]](#)

A four-component array containing the values to fill the unordered-access view resource with.

`NumRects`

Type: [UINT](#)

The number of rectangles in the array that the `pRects` parameter specifies.

`pRects`

Type: [const D3D12_RECT*](#)

An array of D3D12_RECT structures for the rectangles in the resource view to clear. If **NULL**, then **ClearUnorderedAccessViewUint** clears the entire resource view.

Return value

None

Remarks

Runtime validation

Validation failure will result in the call to [ID3D12GraphicsCommandList::Close](#) returning **E_INVALIDARG**.

Debug layer

The debug layer will issue errors if the input values are outside of a normalized range.

The debug layer will issue an error if the subresources referenced by the view are not in the appropriate state. For **ClearUnorderedAccessViewUint**, the state must be [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#).

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::Close method

5/27/2020 • 4 minutes to read • [Edit Online](#)

Indicates that recording to the command list has finished.

Syntax

```
HRESULT Close();
```

Parameters

This method has no parameters.

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:

```
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;
```

```
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<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)));
}

// Create the pipeline state, which includes compiling and loading shaders.
{
    ComPtr<ID3DBlob> vertexShader;
    ComPtr<ID3DBlob> 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();
}
}

```

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::CopyBufferRegion method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void CopyBufferRegion(  
    ID3D12Resource *pDstBuffer,  
    UINT64        DstOffset,  
    ID3D12Resource *pSrcBuffer,  
    UINT64        SrcOffset,  
    UINT64        NumBytes  
) ;
```

Parameters

pDstBuffer

Type: [ID3D12Resource*](#)

Specifies the destination [ID3D12Resource](#).

DstOffset

Type: [UINT64](#)

Specifies a [UINT64](#) offset (in bytes) into the destination resource.

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:

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CopyTextureRegion](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::CopyResource method

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

Syntax

```
void CopyResource(  
    ID3D12Resource *pDstResource,  
    ID3D12Resource *pSrcResource  
)
```

Parameters

pDstResource

Type: [ID3D12Resource*](#)

A pointer to the [ID3D12Resource](#) interface that represents the destination resource.

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 may be used to initialize resources which alias the same heap memory. See [CreatePlacedResource](#) for more details.

Debug layer

The debug layer will issue an error if the source subresource is not in the [D3D12_RESOURCE_STATE_COPY_SOURCE](#) state.

The debug layer will issue an error if the destination subresource is not in the [D3D12_RESOURCE_STATE_COPY_DEST](#) state.

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 have identical dimensions (including width, height, depth, and size as appropriate).
- 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 an

`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. For more info, see [Format Conversion using Direct3D 10.1](#).

- 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. For more info, see [Format Conversion using Direct3D 10.1](#).

You can't use an **Immutable** resource as a destination. You can use a **depth-stencil** resource as either a source or a destination provided that the feature level is `D3D_FEATURE_LEVEL_10_1` or greater. For feature levels `9_x`, resources created with the `D3D12_RESOURCE_MISC_ALLOW_DEPTH_STENCIL` flag can only be used as a source for **CopyResource**. 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.

Examples

The [D3D12HeterogeneousMultiadapter](#) sample uses **CopyResource** in the following way:

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::CopyTextureRegion method

6/25/2020 • 4 minutes to read • [Edit Online](#)

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

```
void CopyTextureRegion(
    const D3D12_TEXTURE_COPY_LOCATION *pDst,
    UINT                               DstX,
    UINT                               DstY,
    UINT                               DstZ,
    const D3D12_TEXTURE_COPY_LOCATION *pSrc,
    const D3D12_BOX                  *pSrcBox
);
```

Parameters

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.

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.

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_FORMATs](#) (identical or from the same type group). For example, a `DXGI_FORMAT_R32G32B32_FLOAT` texture can be copied to an `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:

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[CopyBufferRegion](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::CopyTiles method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Copies tiles from buffer to tiled resource or vice versa.

Syntax

```
void CopyTiles(
    ID3D12Resource                  *pTiledResource,
    const D3D12_TILED_RESOURCE_COORDINATE *pTileRegionStartCoordinate,
    const D3D12_TILE_REGION_SIZE        *pTileRegionSize,
    ID3D12Resource                   *pBuffer,
    UINT64                           BufferStartOffsetInBytes,
    D3D12_TILE_COPY_FLAGS            Flags
);
```

Parameters

`pTiledResource`

Type: `ID3D12Resource*`

A pointer to a tiled resource.

`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.

`pTileRegionSize`

Type: `const D3D12_TILE_REGION_SIZE*`

A pointer to a `D3D12_TILE_REGION_SIZE` structure that describes the size of the tiled region.

`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 one tile, 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Tiled resources](#)

ID3D12GraphicsCommandList::DiscardResource method

5/15/2020 • 2 minutes to read • [Edit Online](#)

Indicates that the contents of a resource don't need to be preserved. The function may re-initialize resource metadata in some cases.

Syntax

```
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

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](#)

ID3D12GraphicsCommandList::Dispatch method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Executes a command list from a thread group.

Syntax

```
void Dispatch(  
    UINT ThreadGroupCountX,  
    UINT ThreadGroupCountY,  
    UINT ThreadGroupCountZ  
) ;
```

Parameters

`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).

`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).

`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:

```

// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::DrawIndexedInstanced method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Draws indexed, instanced primitives.

Syntax

```
void DrawIndexedInstanced(
    UINT IndexCountPerInstance,
    UINT InstanceCount,
    UINT StartIndexLocation,
    INT BaseVertexLocation,
    UINT StartInstanceLocation
);
```

Parameters

IndexCountPerInstance

Type: [UINT](#)

Number of indices read from the index buffer for each instance.

InstanceCount

Type: [UINT](#)

Number of instances to draw.

StartIndexLocation

Type: [UINT](#)

The location of the first index read by the GPU from the index buffer.

BaseVertexLocation

Type: [INT](#)

A value added to each index before reading a vertex from the vertex buffer.

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:

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::DrawInstanced method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Draws non-indexed, instanced primitives.

Syntax

```
void DrawInstanced(  
    UINT VertexCountPerInstance,  
    UINT InstanceCount,  
    UINT StartVertexLocation,  
    UINT StartInstanceLocation  
) ;
```

Parameters

`VertexCountPerInstance`

Type: [UINT](#)

Number of vertices to draw.

`InstanceCount`

Type: [UINT](#)

Number of instances to draw.

`StartVertexLocation`

Type: [UINT](#)

Index of the first vertex.

`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:

```
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;
```

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::EndEvent method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void EndEvent();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::EndQuery method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Ends a running query.

Syntax

```
void EndQuery(  
    ID3D12QueryHeap  *pQueryHeap,  
    D3D12_QUERY_TYPE Type,  
    UINT             Index  
) ;
```

Parameters

pQueryHeap

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the query.

Type

Type: [D3D12_QUERY_TYPE](#)

Specifies one member of [D3D12_QUERY_TYPE](#).

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:

```
// 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()));
```

```

    m_commandList->Reset(m_commandAllocator, 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ExecuteBundle method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Executes a bundle.

Syntax

```
void ExecuteBundle(  
    ID3D12GraphicsCommandList *pCommandList  
) ;
```

Parameters

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:

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ExecuteIndirect method

5/27/2020 • 4 minutes to read • [Edit Online](#)

Apps perform indirect draws/dispatches using the `ExecuteIndirect` method.

Syntax

```
void ExecuteIndirect(
    ID3D12CommandSignature *pCommandSignature,
    UINT                  MaxCommandCount,
    ID3D12Resource        *pArgumentBuffer,
    UINT64                ArgumentBufferOffset,
    ID3D12Resource        *pCountBuffer,
    UINT64                CountBufferOffset
);
```

Parameters

`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.

`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.

`pArgumentBuffer`

Type: [ID3D12Resource*](#)

Specifies one or more [ID3D12Resource](#) objects, containing the command arguments.

`ArgumentBufferOffset`

Type: `UINT64`

Specifies an offset into `pArgumentBuffer` to identify the first command argument.

`pCountBuffer`

Type: [ID3D12Resource*](#)

Specifies a pointer to a [ID3D12Resource](#).

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*:

```
// 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*:

```
// 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:

```
// 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."

```
// 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_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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Indirect Drawing](#)

ID3D12GraphicsCommandList::IASetIndexBuffer method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the view for the index buffer.

Syntax

```
void IASetIndexBuffer(  
    const D3D12_INDEX_BUFFER_VIEW *pView  
) ;
```

Parameters

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:

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

DLL	D3d12.dll
-----	-----------

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::IASetPrimitiveTopology method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Bind information about the primitive type, and data order that describes input data for the input assembler stage.

Syntax

```
void IASetPrimitiveTopology(  
    D3D12_PRIMITIVE_TOPOLOGY PrimitiveTopology  
)
```

Parameters

`PrimitiveTopology`

Type: `D3D12_PRIMITIVE_TOPOLOGY`

The type of primitive and ordering of the primitive data (see [D3D_PRIMITIVE_TOPOLOGY](#)).

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[D3D12_PRIMITIVE_TOPOLOGY_TYPE](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::IASetVertexBuffers method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a CPU descriptor handle for the vertex buffers.

Syntax

```
void IASetVertexBuffers(  
    UINT StartSlot,  
    UINT NumViews,  
    const D3D12_VERTEX_BUFFER_VIEW *pViews  
)
```

Parameters

StartSlot

Type: **UINT**

Index into the device's zero-based array to begin setting vertex buffers.

NumViews

Type: **UINT**

The number of views in the *pViews* array.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[IASetIndexBuffer](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::OMSetBlendFactor method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void OMSetBlendFactor(  
    const FLOAT [4] BlendFactor  
>;
```

Parameters

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::OMSetRenderTargets method

6/25/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void OMSetRenderTargets(  
    UINT NumRenderTargetDescriptors,  
    const D3D12_CPU_DESCRIPTOR_HANDLE *pRenderTargetDescriptors,  
    BOOL RTsSingleHandleToDescriptorRange,  
    const D3D12_CPU_DESCRIPTOR_HANDLE *pDepthStencilDescriptor  
>;
```

Parameters

`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.

`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.

`RTsSingleHandleToDescriptorRange`

Type: `BOOL`

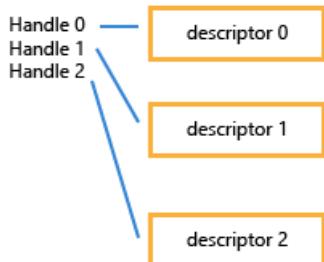
`True` means the handle passed in is the pointer to a contiguous range of `NumRenderTargetDescriptors` descriptors. This case is useful if the set of descriptors to bind already happens to be contiguous in memory (so all that's needed is a handle to the first one). For example, if `NumRenderTargetDescriptors` 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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::OMSetStencilRef method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the reference value for depth stencil tests.

Syntax

```
void OMSetStencilRef(  
    UINT StencilRef  
)
```

Parameters

`StencilRef`

Type: `UINT`

Reference value to perform against when doing a depth-stencil test.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::Reset method

5/27/2020 • 3 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT Reset(  
    ID3D12CommandAllocator *pAllocator,  
    ID3D12PipelineState     *pInitialState  
) ;
```

Parameters

pAllocator

Type: [ID3D12CommandAllocator*](#)

A pointer to the [ID3D12CommandAllocator](#) object that the device creates command lists from.

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. A typical pattern is to submit a command list and then immediately reset it to reuse the allocated memory for another

command list.

You can use **Reset** for both direct command lists and bundles.

The command allocator that **Reset** takes as input can be associated with no more than one recording command list at a time. 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:

```
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;
```

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12CommandAllocator::Reset](#)

[ID3D12Device::CreateCommandList](#)

ID3D12GraphicsCommandList

ID3D12GraphicsCommandList::ResolveQueryData method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void ResolveQueryData(
    ID3D12QueryHeap  *pQueryHeap,
    D3D12_QUERY_TYPE Type,
    UINT             StartIndex,
    UINT             NumQueries,
    ID3D12Resource   *pDestinationBuffer,
    UINT64           AlignedDestinationBufferOffset
);
```

Parameters

pQueryHeap

Type: [ID3D12QueryHeap*](#)

Specifies the [ID3D12QueryHeap](#) containing the queries to resolve.

Type

Type: [D3D12_QUERY_TYPE](#)

Specifies the type of query, one member of [D3D12_QUERY_TYPE](#).

StartIndex

Type: [UINT](#)

Specifies an index of the first query to resolve.

NumQueries

Type: [UINT](#)

Specifies the number of queries to resolve.

pDestinationBuffer

Type: [ID3D12Resource*](#)

Specifies an [ID3D12Resource](#) destination buffer, which must be in the state [D3D12_RESOURCE_STATE_COPY_DEST](#).

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 which writes query data into a destination buffer. Query data is written contiguously to the destination buffer, and the parameter.

Binary occlusion queries write 64-bits per query. The least significant bit is either 0 or 1. The rest of the bits are 0.

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.

Examples

The [D3D12PredicationQueries](#) sample uses **ID3D12GraphicsCommandList::ResolveQueryData** as follows:

```
// 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.
    r
```

```

    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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::ResolveSubresource method

6/26/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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:

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Subresources](#)

ID3D12GraphicsCommandList::ResourceBarrier method

5/27/2020 • 5 minutes to read • [Edit Online](#)

Notifies the driver that it needs to synchronize multiple accesses to resources.

Syntax

```
void ResourceBarrier(  
    UINT                 NumBarriers,  
    const D3D12_RESOURCE_BARRIER *pBarriers  
>;
```

Parameters

NumBarriers

Type: **UINT**

The number of submitted barrier descriptions.

pBarriers

Type: **const D3D12_RESOURCE_BARRIER***

Pointer to an array of barrier descriptions.

Return value

None

Remarks

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 D3D11 debug layer would issue warnings for [ID3D11DeviceContext2::TiledResourceBarrier](#).
- 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:

```
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;
```

```

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

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](#)

ID3D12GraphicsCommandList::RSSetScissorRects method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Binds an array of scissor rectangles to the rasterizer stage.

Syntax

```
void RSSetScissorRects(  
    UINT           NumRects,  
    const D3D12_RECT *pRects  
>;
```

Parameters

`NumRects`

Type: `UINT`

The number of scissor rectangles to bind.

`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:

```

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;

```

```

// 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

Target Platform	Windows
Header	d3d12.h

Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::RSSetViewports method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Bind an array of viewports to the rasterizer stage of the pipeline.

Syntax

```
void RSSetViewports(  
    UINT           NumViewports,  
    const D3D12_VIEWPORT *pViewports  
)
```

Parameters

NumViewports

Type: **UINT**

Number of viewports to bind. The range of valid values is (0, **D3D12_VIEWPORT_AND_SCISSORRECT_OBJECT_COUNT_PER_PIPELINE**).

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.

Note Even though you specify float values to the members of the **D3D12_VIEWPORT** structure for the *pViewports* array in a call to **RSSetViewports** for **feature levels** 9_x, **RSSetViewports** uses DWORDs internally. Because of this behavior, when you use a negative top left corner for the viewport, the call to **RSSetViewports** for feature levels 9_x fails. This failure occurs because **RSSetViewports** for 9_x casts the floating point values into unsigned integers without validation, which results in integer overflow.

Examples

The [D3D12HelloTriangle](#) sample uses **ID3D12GraphicsCommandList::RSSetViewports** as follows:

```

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;

```

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRoot32BitConstant method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a constant in the compute root signature.

Syntax

```
void SetComputeRoot32BitConstant(  
    UINT RootParameterIndex,  
    UINT SrcData,  
    UINT DestOffsetIn32BitValues  
>;
```

Parameters

RootParameterIndex

Type: **UINT**

The slot number for binding.

SrcData

Type: **UINT**

The source data for the constant to set.

DestOffsetIn32BitValues

Type: **UINT**

The offset, in 32-bit values, to set the constant in the root signature.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRoot32BitConstants method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a group of constants in the compute root signature.

Syntax

```
void SetComputeRoot32BitConstants(  
    UINT      RootParameterIndex,  
    UINT      Num32BitValuesToSet,  
    const void *pSrcData,  
    UINT      DestOffsetIn32BitValues  
) ;
```

Parameters

RootParameterIndex

Type: [UINT](#)

The slot number for binding.

Num32BitValuesToSet

Type: [UINT](#)

The number of constants to set in the root signature.

pSrcData

Type: [const void*](#)

The source data for the group of constants to set.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRootConstantBufferView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetComputeRootConstantBufferView(  
    UINT             RootParameterIndex,  
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation  
>;
```

Parameters

RootParameterIndex

Type: **UINT**

The slot number for binding.

BufferLocation

Type: **D3D12_GPU_VIRTUAL_ADDRESS**

Specifies the D3D12_GPU_VIRTUAL_ADDRESS of the constant buffer.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRootDescriptorTable method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a descriptor table into the compute root signature.

Syntax

```
void SetComputeRootDescriptorTable(  
    UINT RootParameterIndex,  
    D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor  
>;
```

Parameters

RootParameterIndex

Type: **UINT**

The slot number for binding.

BaseDescriptor

Type: **D3D12_GPU_DESCRIPTOR_HANDLE**

A GPU_descriptor_handle object for the base descriptor to set.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRootShaderResourceView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetComputeRootShaderResourceView(  
    UINT             RootParameterIndex,  
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation  
)
```

Parameters

RootParameterIndex

Type: **UINT**

The slot number for binding.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRootSignature method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the layout of the compute root signature.

Syntax

```
void SetComputeRootSignature(  
    ID3D12RootSignature *pRootSignature  
) ;
```

Parameters

`pRootSignature`

Type: [ID3D12RootSignature*](#)

A pointer to the [ID3D12RootSignature](#) object.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetComputeRootUnorderedAccessView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetComputeRootUnorderedAccessView(  
    UINT RootParameterIndex,  
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation  
>;
```

Parameters

RootParameterIndex

Type: [UINT](#)

The slot number for binding.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetDescriptorHeaps method

5/21/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

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.

Examples

The [D3D12Bundles](#) sample uses **ID3D12GraphicsCommandList::SetDescriptorHeaps** as follows:

```

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[Descriptor Heaps](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstant method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a constant in the graphics root signature.

Syntax

```
void SetGraphicsRoot32BitConstant(  
    UINT RootParameterIndex,  
    UINT SrcData,  
    UINT DestOffsetIn32BitValues  
>;
```

Parameters

`RootParameterIndex`

Type: [UINT](#)

The slot number for binding.

`SrcData`

Type: [UINT](#)

The source data for the constant to set.

`DestOffsetIn32BitValues`

Type: [UINT](#)

The offset, in 32-bit values, to set the constant in the root signature.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRoot32BitConstants method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a group of constants in the graphics root signature.

Syntax

```
void SetGraphicsRoot32BitConstants(  
    UINT      RootParameterIndex,  
    UINT      Num32BitValuesToSet,  
    const void *pSrcData,  
    UINT      DestOffsetIn32BitValues  
>;
```

Parameters

RootParameterIndex

Type: **UINT**

The slot number for binding.

Num32BitValuesToSet

Type: **UINT**

The number of constants to set in the root signature.

pSrcData

Type: **const void***

The source data for the group of constants to set.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRootConstantBufferView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetGraphicsRootConstantBufferView(  
    UINT             RootParameterIndex,  
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation  
>;
```

Parameters

RootParameterIndex

Type: [UINT](#)

The slot number for binding.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRootDescriptorTable method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a descriptor table into the graphics root signature.

Syntax

```
void SetGraphicsRootDescriptorTable(
    UINT                    RootParameterIndex,
    D3D12_GPU_DESCRIPTOR_HANDLE BaseDescriptor
);
```

Parameters

`RootParameterIndex`

Type: [UINT](#)

The slot number for binding.

`BaseDescriptor`

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A GPU_descriptor_handle object for the base descriptor to set.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRootShaderResourceView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetGraphicsRootShaderResourceView(
    UINT             RootParameterIndex,
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation
);
```

Parameters

RootParameterIndex

Type: [UINT](#)

The slot number for binding.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRootSignature method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the layout of the graphics root signature.

Syntax

```
void SetGraphicsRootSignature(  
    ID3D12RootSignature *pRootSignature  
>;
```

Parameters

pRootSignature

Type: [ID3D12RootSignature*](#)

A pointer to the [ID3D12RootSignature](#) object.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetGraphicsRootUnorderedAccessView method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetGraphicsRootUnorderedAccessView(  
    UINT RootParameterIndex,  
    D3D12_GPU_VIRTUAL_ADDRESS BufferLocation  
>;
```

Parameters

RootParameterIndex

Type: [UINT](#)

The slot number for binding.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetMarker method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void SetMarker(  
    UINT      Metadata,  
    const void *pData,  
    UINT      Size  
) ;
```

Parameters

Metadata

Type: **UINT**

Internal.

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

Target Platform	Windows
Header	d3d12.h

Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetPipelineState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets all shaders and programs most of the fixed-function state of the graphics processing unit (GPU) pipeline.

Syntax

```
void SetPipelineState(  
    ID3D12PipelineState *pPipelineState  
)
```

Parameters

`pPipelineState`

Type: [ID3D12PipelineState*](#)

Pointer to the [ID3D12PipelineState](#) containing the pipeline state data.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList::SetPredication method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a rendering predicate.

Syntax

```
void SetPredication(
    ID3D12Resource     *pBuffer,
    UINT64              AlignedBufferOffset,
    D3D12_PREDICATION_OP Operation
);
```

Parameters

`pBuffer`

Type: [ID3D12Resource*](#)

The buffer, as an [ID3D12Resource](#).

`AlignedBufferOffset`

Type: [UINT64](#)

The aligned buffer offset, as a [UINT64](#).

`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. However, some predicates are only hints, so they may not actually prevent operations from being performed.

Unlike Direct3D 11, in Direct3D 12 predication state is not inherited by direct command lists. 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:

```
// 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
    }
}
```

```

    // 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

[Predication queries walk-through](#)

ID3D12GraphicsCommandList::SOSetTargets method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets the stream output buffer views.

Syntax

```
void SOSetTargets(
    UINT StartSlot,
    UINT NumViews,
    const D3D12_STREAM_OUTPUT_BUFFER_VIEW *pViews
);
```

Parameters

StartSlot

Type: **UINT**

Index into the device's zero-based array to begin setting stream output buffers.

NumViews

Type: **UINT**

The number of entries in the *pViews* array.

pViews

Type: **const D3D12_STREAM_OUTPUT_BUFFER_VIEW***

Specifies an array of **D3D12_STREAM_OUTPUT_BUFFER_VIEW** structures.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList1 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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 targeting 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](#)

Methods

The [ID3D12GraphicsCommandList1](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12GraphicsCommandList1::AtomicCopyBufferUINT	Atomically copies a primary data element of type <code>UINT</code> from one resource to another, along with optional dependent resources.
ID3D12GraphicsCommandList1::AtomicCopyBufferUINT64	Atomically copies a primary data element of type <code>UINT64</code> 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

Target Platform	Windows
-----------------	---------

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

See also

[Core Interfaces](#)

[ID3D12GraphicsCommandList](#)

ID3D12GraphicsCommandList1::AtomicCopyBufferUINT method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void AtomicCopyBufferUINT(
    ID3D12Resource                 *pDstBuffer,
    UINT64                          DstOffset,
    ID3D12Resource                 *pSrcBuffer,
    UINT64                          SrcOffset,
    UINT                           Dependencies,
    ID3D12Resource                 * const *ppDependentResources,
    const D3D12_SUBRESOURCE_RANGE_UINT64 *pDependentSubresourceRanges
);
```

Parameters

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.

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.

ppDependentResources

Type: **ID3D12Resource***

SAL: *In_reads(Dependencies)*

An array of resources that contain the dependent elements of the data payload.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList1::AtomicCopyBufferUINT64 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void AtomicCopyBufferUINT64(
    ID3D12Resource           *pDstBuffer,
    UINT64                   DstOffset,
    ID3D12Resource           *pSrcBuffer,
    UINT64                   SrcOffset,
    UINT                      Dependencies,
    ID3D12Resource           * const *ppDependentResources,
    const D3D12_SUBRESOURCE_RANGE_UINT64 *pDependentSubresourceRanges
);
```

Parameters

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.

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.

ppDependentResources

Type: **ID3D12Resource***

SAL: *In_reads(Dependencies)*

An array of resources that contain the dependent elements of the data payload.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList1::OMSetDepthBounds method

5/27/2020 • 2 minutes to read • [Edit Online](#)

This method enables you to change the depth bounds dynamically.

Syntax

```
void OMSetDepthBounds(  
    FLOAT Min,  
    FLOAT Max  
) ;
```

Parameters

Min

Type: **FLOAT**

SAL: *In*

Specifies the minimum depth bounds. The default value is 0. NaN values silently convert to 0.

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_OPTIONS1](#) structure.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList1::ResolveSubresourceRegion method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void ResolveSubresourceRegion(
    ID3D12Resource     *pDstResource,
    UINT                DstSubresource,
    UINT                DstX,
    UINT                DstY,
    ID3D12Resource     *pSrcResource,
    UINT                SrcSubresource,
    D3D12_RECT          *pSrcRect,
    DXGI_FORMAT          Format,
    D3D12_RESOLVE_MODE ResolveMode
);
```

Parameters

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.

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.

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.

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.

`pSrcResource`

Type: `ID3D12Resource*`

SAL: `In`

Source resource. Must be multisampled or compressed.

`SrcSubresource`

Type: `UINT`

SAL: `In`

A zero-based index that identifies the source subresource.

`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.

`Format`

Type: `DXGI_FORMAT`

SAL: `In`

A `DXGI_FORMAT` that specifies how the source and destination resource formats are consolidated.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList1::SetSamplePositions method

5/27/2020 • 7 minutes to read • [Edit Online](#)

This method configures the sample positions used by subsequent draw, copy, resolve, and similar operations.

Syntax

```
void SetSamplePositions(  
    UINT             NumSamplesPerPixel,  
    UINT             NumPixels,  
    D3D12_SAMPLE_POSITION *pSamplePositions  
)
```

Parameters

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.

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.

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 prioritiy. 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 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 postions 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

Target Platform	Windows
Header	d3d12.h

Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList1::SetViewInstanceMask method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Set a mask that controls which view instances are enabled for subsequent draws.

Syntax

```
void SetViewInstanceMask(  
    UINT Mask  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList2 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The [ID3D12GraphicsCommandList2](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12GraphicsCommandList2::WriteBufferImmediate	Writes a number of 32-bit immediate values to the specified buffer locations directly from the command stream.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

Core Interfaces

[ID3D12GraphicsCommandList](#)

[ID3D12GraphicsCommandList1](#)

ID3D12GraphicsCommandList2::WriteBufferImmediate method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Writes a number of 32-bit immediate values to the specified buffer locations directly from the command stream.

Syntax

```
void WriteBufferImmediate(  
    UINT                                     Count,  
    const D3D12_WRITEBUFFERIMMEDIATE_PARAMETER *pParams,  
    const D3D12_WRITEBUFFERIMMEDIATE_MODE      *pModes  
>;
```

Parameters

Count

The number of [D3D12_WRITEBUFFERIMMEDIATE_PARAMETER](#) structures that are pointed to by *pParams* and *pModes*.

pParams

The address of an array containing a number of [D3D12_WRITEBUFFERIMMEDIATE_PARAMETER](#) structures equal to *Count*.

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

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](#)

ID3D12GraphicsCommandList3 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Encapsulates a list of graphics commands for rendering.

Inheritance

The ID3D12GraphicsCommandList3 interface inherits from [ID3D12GraphicsCommandList2](#).

ID3D12GraphicsCommandList3 also has these types of members:

- [Methods](#)

Methods

The ID3D12GraphicsCommandList3 interface has these methods.

METHOD	DESCRIPTION
ID3D12GraphicsCommandList3::SetProtectedResourceSession	Specifies whether or not protected resources can be accessed by subsequent commands in the command list.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList2](#)

ID3D12GraphicsCommandList3::SetProtectedResourceSession method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void SetProtectedResourceSession(  
    ID3D12ProtectedResourceSession *pProtectedResourceSession  
>);
```

Parameters

`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

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](#)

ID3D12GraphicsCommandList4 interface

6/25/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12GraphicsCommandList4** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList3](#)

ID3D12GraphicsCommandList4::BeginRenderPass method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void BeginRenderPass(  
    UINT NumRenderTargets,  
    const D3D12_RENDER_PASS_RENDER_TARGET_DESC *pRenderTargets,  
    const D3D12_RENDER_PASS_DEPTH_STENCIL_DESC *pDepthStencil,  
    D3D12_RENDER_PASS_FLAGS Flags  
) ;
```

Parameters

NumRenderTargets

A **UINT**. The number of render targets being bound.

pRenderTargets

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

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](#)

ID3D12GraphicsCommandList4::BuildRaytracingAccelerationStructure method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Performs a raytracing acceleration structure build on the GPU and optionally outputs post-build information immediately after the build.

Syntax

```
void BuildRaytracingAccelerationStructure(
    const D3D12_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_DESC* pDesc,
    UINT NumPostbuildInfoDescs,
    const D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC* pPostbuildInfoDescs
);
```

Parameters

pDesc

Description of the acceleration structure to build.

NumPostbuildInfoDescs

Size of the *pPostbuildInfoDescs* array. Set to 0 if no post-build info is needed.

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

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](#)

ID3D12GraphicsCommandList4::CopyRaytracingAccelerationStructure method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void CopyRaytracingAccelerationStructure(
    D3D12_GPU_VIRTUAL_ADDRESS DestAccelerationStructureData,
    D3D12_GPU_VIRTUAL_ADDRESS SourceAccelerationStructureData,
    D3D12_RAYTRACING_ACCELERATION_STRUCTURE_COPY_MODE Mode
);
```

Parameters

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](#).

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](#).

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

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](#)

ID3D12GraphicsCommandList4::DispatchRays method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Launch the threads of a ray generation shader.

Syntax

```
void DispatchRays(  
    const D3D12_DISPATCH_RAYS_DESC *pDesc  
)
```

Parameters

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 \times height \times depth $\leq 2^{30}$. Exceeding this produces undefined behavior. If any grid dimension is 0, no threads are launched.

Requirements

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](#)

ID3D12GraphicsCommandList4::EmitRaytracingAccelerationStructurePostbuildInfo method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void EmitRaytracingAccelerationStructurePostbuildInfo(
    const D3D12_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_DESC *pDesc,
    UINT NumSourceAccelerationStructures,
    const D3D12_GPU_VIRTUAL_ADDRESS *pSourceAccelerationStructureData
);
```

Parameters

pDesc

Description of pos-tbuild information to generate.

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.

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

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](#)

ID3D12GraphicsCommandList4::EndRenderPass method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Marks the ending of a render pass.

Syntax

```
void EndRenderPass();
```

Parameters

This method has no parameters.

Return value

None

Requirements

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](#)

ID3D12GraphicsCommandList4::ExecuteMetaCommand method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void ExecuteMetaCommand(  
    ID3D12MetaCommand *pMetaCommand,  
    const void        *pExecutionParametersData,  
    SIZE_T            ExecutionParametersDataSizeInBytes  
) ;
```

Parameters

`pMetaCommand`

A pointer to an [ID3D12MetaCommand](#) representing the meta command to initialize.

`pExecutionParametersData`

An optional pointer to a constant structure containing the values of the parameters for executing the meta command.

`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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList4](#)

ID3D12GraphicsCommandList4::InitializeMetaCommand method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void InitializeMetaCommand(
    ID3D12MetaCommand *pMetaCommand,
    const void        *pInitializationParametersData,
    SIZE_T            InitializationParametersDataSizeInBytes
);
```

Parameters

`pMetaCommand`

A pointer to an [ID3D12MetaCommand](#) representing the meta command to initialize.

`pInitializationParametersData`

An optional pointer to a constant structure containing the values of the parameters for initializing the meta command.

`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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12GraphicsCommandList4](#)

ID3D12GraphicsCommandList4::SetPipelineState1 method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets a state object on the command list.

Syntax

```
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

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](#)

ID3D12GraphicsCommandList5 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12GraphicsCommandList5::RSSetShadingRate	
ID3D12GraphicsCommandList5::RSSetShadingRateImage	

Requirements

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

See also

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

ID3D12GraphicsCommandList5::RSSetShadingRate method

1/11/2020 • 2 minutes to read • [Edit Online](#)

Sets the base shading rate, and combiners, for variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

```
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](#).

Return value

None

Requirements

Header

d3d12.h

See also

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

ID3D12GraphicsCommandList5::RSSetShadingRateImage method

1/11/2020 • 2 minutes to read • [Edit Online](#)

Sets the screen-space shading-rate image for variable-rate shading (VRS). For more info, see [Variable-rate shading \(VRS\)](#).

Syntax

```
void RSSetShadingRateImage(  
    ID3D12Resource *shadingRateImage  
)
```

Parameters

shadingRateImage

Type: [ID3D12Resource*](#)

An optional pointer to an [ID3D12Resource](#) representing a screen-space shading-rate image.

Return value

None

Requirements

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

See also

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

ID3D12Heap interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The [ID3D12Heap](#) interface has these methods.

METHOD	DESCRIPTION
ID3D12Heap::GetDesc	Gets the heap description.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

[Memory Management in Direct3D 12](#)

ID3D12Heap::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the heap description.

Syntax

```
D3D12_HEAP_DESC GetDesc();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_HEAP_DESC](#)

Returns the [D3D12_HEAP_DESC](#) structure that describes the heap.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Heap](#)

ID3D12LifetimeOwner interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12LifetimeOwner::LifetimeStateUpdated	Called when the lifetime state of a lifetime-tracked object changes.

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12LifetimeOwner::LifetimeStateUpdated method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Called when the lifetime state of a lifetime-tracked object changes.

Syntax

```
void LifetimeStateUpdated(  
    D3D12_LIFETIME_STATE NewState  
>);
```

Parameters

NewState

Type: [D3D12_LIFETIME_STATE](#)

The new state.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12LifetimeTracker interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12LifetimeTracker::DestroyOwnedObject	Destroys a lifetime-tracked object.

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12LifetimeTracker::DestroyOwnedObject method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Destroys a lifetime-tracked object.

Syntax

```
HRESULT DestroyOwnedObject(  
    ID3D12DeviceChild *pObject  
>;
```

Parameters

pObject

Type: [ID3D12DeviceChild*](#)

A pointer to an [ID3D12DeviceChild](#) interface representing the lifetime-tracked object.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12.h

ID3D12MetaCommand interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12MetaCommand** interface has these methods.

METHOD	DESCRIPTION
ID3D12MetaCommand::GetRequiredParameterResourceSize	Retrieves the amount of memory required for the specified runtime parameter resource for a meta command, for the specified stage.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Pageable](#)

ID3D12MetaCommand::GetRequiredParameterResourceSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetRequiredParameterResourceSize(  
    D3D12_META_COMMAND_PARAMETER_STAGE Stage,  
    UINT ParameterIndex  
) ;
```

Parameters

Stage

Type: [D3D12_META_COMMAND_PARAMETER_STAGE](#)

A [D3D12_META_COMMAND_PARAMETER_STAGE](#) specifying the stage to which the parameter belongs.

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

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12MetaCommand](#)

ID3D12Object interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12Object** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

ID3D12Object::GetPrivateData method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets application-defined data from a device object.

Syntax

```
HRESULT GetPrivateData(  
    REFGUID guid,  
    UINT     *pDataSize,  
    void     *pData  
>;
```

Parameters

guid

Type: [REFGUID](#)

The GUID that is associated with the data.

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.

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](#).

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Object](#)

ID3D12Object::SetName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Associates a name with the device object. This name is for use in debug diagnostics and tools.

Syntax

```
HRESULT SetName(  
    LPCWSTR Name  
)
```

Parameters

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 **WKP DID_D3DDebugObjectNameW**. 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 **WKP DID_D3DDebugObjectName** GUID directly instead.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[Direct3D 12 Programming Environment Setup](#)

ID3D12Object

ID3D12Object::SetPrivateData method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT SetPrivateData(  
    REFGUID    guid,  
    UINT       DataSize,  
    const void *pData  
)
```

Parameters

guid

Type: [REFGUID](#)

The GUID to associate with the data.

DataSize

Type: [UINT](#)

The size in bytes of the data.

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 **WKPDID_D3DDescribeObjectName** with an ASCII name, call [ID3D12Object::SetName](#) with a UNICODE name.

Requirements

Target Platform	Windows
Header	d3d12.h

Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Object](#)

ID3D12Object::SetPrivateDataInterface method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Associates an [IUnknown](#)-derived interface with the device object and associates that interface with an application-defined GUID.

Syntax

```
HRESULT SetPrivateDataInterface(  
    REFGUID      guid,  
    const IUnknown *pData  
)
```

Parameters

guid

Type: [REFGUID](#)

The GUID to associate with the interface.

pData

Type: [const IUnknown*](#)

A pointer to the [IUnknown](#)-derived interface to be associated with the device object.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Object](#)

ID3D12Pageable interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

Methods

The **ID3D12Pageable** interface has these methods.

METHOD	DESCRIPTION

Remarks

For more details, refer to [Memory Management in Direct3D 12](#) and the [MakeResident](#) method reference.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[Creating Descriptor Heaps](#)

[ID3D12DeviceChild](#)

ID3D12PipelineLibrary interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12PipelineLibrary** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12DeviceChild](#)

Root Signature Version 1.1

ID3D12PipelineLibrary::GetSerializedSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Returns the amount of memory required to serialize the current contents of the database.

Syntax

```
SIZE_T GetSerializedSize();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

ID3D12PipelineLibrary::LoadComputePipeline method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT LoadComputePipeline(
    LPCWSTR                pName,
    const D3D12_COMPUTE_PIPELINE_STATE_DESC *pDesc,
    REFIID                  riid,
    void                    **ppPipelineState
);
```

Parameters

pName

Type: **LPCWSTR**

The unique name of the PSO.

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.

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

System Requirements	
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

ID3D12PipelineLibrary::LoadGraphicsPipeline method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the requested PSO from the library.

Syntax

```
HRESULT LoadGraphicsPipeline(
    LPCWSTR                      pName,
    const D3D12_GRAPHICS_PIPELINE_STATE_DESC *pDesc,
    REFIID                         riid,
    void                           **ppPipelineState
);
```

Parameters

pName

Type: **LPCWSTR**

The unique name of the PSO.

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.

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

System Requirements	
Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

ID3D12PipelineLibrary::Serialize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT Serialize(  
    void    *pData,  
    SIZE_T  DataSizeInBytes  
) ;
```

Parameters

`pData`

Type: `void*`

Specifies a pointer to the data. This memory must be readable and writeable 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.

`DataSizeInBytes`

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary](#)

ID3D12PipelineLibrary::StorePipeline method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Adds the input PSO to an internal database with the corresponding name.

Syntax

```
HRESULT StorePipeline(  
    LPCWSTR          pName,  
    ID3D12PipelineState *pPipeline  
)
```

Parameters

pName

Type: **LPCWSTR**

Specifies a unique name for the library. Overwriting is not supported.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

ID3D12PipelineLibrary

ID3D12PipelineLibrary1 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12PipelineLibrary1** interface has these methods.

METHOD	DESCRIPTION
ID3D12PipelineLibrary1::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.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12PipelineLibrary](#)

ID3D12PipelineLibrary1::LoadPipeline method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT LoadPipeline(  
    LPCWSTR pName,  
    const D3D12_PIPELINE_STATE_STREAM_DESC *pDesc,  
    REFIID riid,  
    void **ppPipelineState  
)
```

Parameters

pName

Type: **LPCWSTR**

SAL: *In*

The unique name of the PSO.

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.

Applications 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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineLibrary1](#)

ID3D12PipelineState interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12PipelineState** interface has these methods.

METHOD	DESCRIPTION
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.

```
// 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.

```
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.

```

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

Target Platform	Windows
Header	d3d12.h

See also

Core Interfaces

ID3D12Pageable

ID3D12PipelineState::GetCachedBlob method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the cached blob representing the pipeline state.

Syntax

```
HRESULT GetCachedBlob(  
    ID3DBlob **ppBlob  
)
```

Parameters

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12PipelineState](#)

ID3D12ProtectedResourceSession interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12ProtectedResourceSession::GetDesc	Retrieves a description of the protected resource session.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12ProtectedSession](#)

ID3D12ProtectedResourceSession::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves a description of the protected resource session.

Syntax

```
D3D12_PROTECTED_RESOURCE_SESSION_DESC GetDesc();
```

Parameters

This method has no parameters.

Return value

A D3D12_PROTECTED_RESOURCE_SESSION_DESC that describes the protected resource session.

Requirements

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](#)

ID3D12ProtectedSession interface

4/22/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12ProtectedSession** interface has these methods.

METHOD	DESCRIPTION
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 ID3D12Fence::GetCompletedValue), and add monitors for changes to its value. This is a read-only fence.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12DeviceChild](#)

ID3D12ProtectedSession::GetSessionStatus method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the status of the protected session.

Syntax

```
D3D12_PROTECTED_SESSION_STATUS GetSessionStatus();
```

Parameters

This method has no parameters.

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

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](#)

ID3D12ProtectedSession::GetStatusFence method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
HRESULT GetStatusFence(  
    REFIID riid,  
    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](#).

`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

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](#)

ID3D12QueryHeap interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Inheritance

The ID3D12QueryHeap interface inherits from the ID3D12Pageable interface.

Methods

The ID3D12QueryHeap interface has these methods.

METHOD	DESCRIPTION

Remarks

For more information, refer to [Queries](#).

Examples

The [D3D12PredicationQueries](#) sample uses ID3D12QueryHeap as follows:

Create a query heap and a query result buffer.

```
// 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;
```

```
// 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)));\n
```

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

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

ID3D12Resource interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12Resource** interface has these methods.

METHOD	DESCRIPTION
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	Invalidates the CPU pointer to the specified subresource in the resource. Unmap also flushes the CPU cache, when necessary, so that GPU reads to this address reflect any modifications made by the CPU.
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12Pageable](#)

ID3D12Resource::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the resource description.

Syntax

```
D3D12_RESOURCE_DESC GetDesc();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_RESOURCE_DESC](#)

A Direct3D 12 resource description structure.

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Resource](#)

ID3D12Resource::GetGPUVirtualAddress method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
D3D12_GPU_VIRTUAL_ADDRESS GetGPUVirtualAddress();
```

Parameters

This method has no parameters.

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:

```
// 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

ID3D12Resource

ID3D12Resource::GetHeapProperties method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT GetHeapProperties(  
    D3D12_HEAP_PROPERTIES *pHeapProperties,  
    D3D12_HEAP_FLAGS      *pHeapFlags  
) ;
```

Parameters

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.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib

DLL	D3d12.dll
-----	-----------

See also

[ID3D12Resource](#)

ID3D12Resource::Map method

5/27/2020 • 5 minutes to read • [Edit Online](#)

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

```
HRESULT Map(  
    UINT             Subresource,  
    const D3D12_RANGE *pReadRange,  
    void            **ppData  
)
```

Parameters

Subresource

Type: **UINT**

Specifies the index number of the subresource.

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**.

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_PAGE_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.

Resources on D3D12_HEAP_TYPE_READBACK heaps do not support persistent map. **Map** and **Unmap** must be called between CPU and GPU accesses to the same memory address on some system architectures, when the page caching behavior is write-back. **Map** and **Unmap** invalidate and flush the last level CPU cache on some ARM systems, to marshal data between the CPU and GPU through memory addresses with write-back behavior.

Advanced Usage Models

Resources on D3D12_HEAP_TYPE_UPLOAD 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 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.

Applications may understand the adapter architectural details and use custom heaps to write optimizations for UMA architectures, multi-engine applications, multi-adapter applications, and other less common scenarios. Persistent map can be used on the custom heap types when the adapter architectures supports it. Heaps with write-combine properties always support persistent map, and heaps with write-back properties support persistent map on non-ARM systems.

Examples

The [D3D12Bundles](#) sample uses `ID3D12Resource::Map` as follows:

Copy triangle data to the vertex buffer.

```
// 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.

```
// 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 'permenantly' 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Resource](#)

[Subresources](#)

[Unmap](#)

ID3D12Resource::ReadFromSubresource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT ReadFromSubresource(  
    void* pDstData,  
    UINT DstRowPitch,  
    UINT DstDepthPitch,  
    UINT SrcSubresource,  
    const D3D12_BOX *pSrcBox  
) ;
```

Parameters

`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.

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

[Subresources](#)

ID3D12Resource::Unmap method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Invalidates the CPU pointer to the specified subresource in the resource. **Unmap** also flushes the CPU cache, when necessary, so that GPU reads to this address reflect any modifications made by the CPU.

Syntax

```
void Unmap(
    UINT           Subresource,
    const D3D12_RANGE *pWrittenRange
);
```

Parameters

Subresource

Type: **UINT**

Specifies the index of the subresource.

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**.

Return value

None

Remarks

Refer to the extensive Remarks and Examples for the [Map](#) method.

Requirements

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Resource](#)

[Map](#)

[Subresources](#)

ID3D12Resource::WriteToSubresource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT WriteToSubresource(  
    UINT             DstSubresource,  
    const D3D12_BOX *pDstBox,  
    const void       *pSrcData,  
    UINT             SrcRowPitch,  
    UINT             SrcDepthPitch  
) ;
```

Parameters

DstSubresource

Type: **UINT**

Specifies the index of the subresource.

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.

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 ($64 / [\text{bytes per pixel}]$) 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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12Resource](#)

[Subresources](#)

ID3D12RootSignature interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

Methods

The ID3D12RootSignature interface has these methods.

METHOD	DESCRIPTION
--------	-------------

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[ID3D12DeviceChild](#)

[Root Signatures](#)

ID3D12RootSignatureDeserializer interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12RootSignatureDeserializer** interface has these methods.

METHOD	DESCRIPTION
ID3D12RootSignatureDeserializer::GetRootSignatureDesc	Gets the layout of the root signature.

Remarks

This interface has been superceded by [ID3D12VersionedRootSignatureDeserializer](#).

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

[Root Signatures](#)

ID3D12RootSignatureDeserializer::GetRootSignatureDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the layout of the root signature.

Syntax

```
const D3D12_ROOT_SIGNATURE_DESC * GetRootSignatureDesc();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12RootSignatureDeserializer](#)

ID3D12StateObject interface

7/1/2020 • 2 minutes to read • [Edit Online](#)

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.

Methods

The ID3D12StateObject interface has these methods.

METHOD	DESCRIPTION
--------	-------------

Requirements

Target Platform	Windows
Header	d3d12.h

See also

[ID3D12Pageable](#)

ID3D12StateObjectProperties interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Provides methods for getting and setting the properties of an [ID3D12StateObject](#).

Inheritance

The **ID3D12StateObjectProperties** interface inherits from the [IUnknown](#) interface.

ID3D12StateObjectProperties also has these types of members:

- [Methods](#)

Methods

The **ID3D12StateObjectProperties** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

ID3D12StateObjectProperties::GetPipelineStackSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the current pipeline stack size.

Syntax

```
UINT64 GetPipelineStackSize();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

ID3D12StateObjectProperties::GetShaderIdentifier method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the unique identifier for a shader that can be used in a shader record.

Syntax

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

ID3D12StateObjectProperties::GetShaderStackSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the amount of stack memory required to invoke a raytracing shader in HLSL.

Syntax

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12StateObjectProperties](#)

ID3D12StateObjectProperties::SetPipelineStackSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Set the current pipeline stack size.

Syntax

```
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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

ID3D12StateObjectProperties

ID3D12Tools interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

This interface is used to configure the runtime for tools such as PIX. Its 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](#)

Methods

The **ID3D12Tools** interface has these methods.

METHOD	DESCRIPTION
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, its 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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

ID3D12Tools::EnableShaderInstrumentation method

5/27/2020 • 2 minutes to read • [Edit Online](#)

This method enables tools such as PIX to instrument shaders.

Syntax

```
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, its 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Tools](#)

ID3D12Tools::ShaderInstrumentationEnabled method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Determines whether shader instrumentation is enabled.

Syntax

```
BOOL ShaderInstrumentationEnabled();
```

Parameters

This method has no parameters.

Return value

Type: **BOOL**

Returns TRUE if shader instrumentation is enabled; otherwise FALSE.

Remarks

Do not use this interface in your application, its 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

Target Platform	Windows
Header	d3d12.h
Library	D3D12.lib
DLL	D3D12.dll

See also

[ID3D12Tools](#)

ID3D12VersionedRootSignatureDeserializer interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12VersionedRootSignatureDeserializer** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12.h

See also

[Core Interfaces](#)

[IUnknown](#)

[Root Signature Version 1.1](#)

[Root Signatures](#)

ID3D12VersionedRootSignatureDeserializer::GetRootSignatureDescAtVersion method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Converts root signature description structures to a requested version.

Syntax

```
HRESULT GetRootSignatureDescAtVersion(
    D3D_ROOT_SIGNATURE_VERSION           convertToVersion,
    const D3D12_VERSIONED_ROOT_SIGNATURE_DESC **ppDesc
);
```

Parameters

`convertToVersion`

Type: [D3D_ROOT_SIGNATURE_VERSION](#)

Specifies the required [D3D_ROOT_SIGNATURE_VERSION](#).

`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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12VersionedRootSignatureDeserializer](#)

[Root Signature Version 1.1](#)

ID3D12VersionedRootSignatureDeserializer::GetUnconvertedRootSignatureDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
const D3D12_VERSIONED_ROOT_SIGNATURE_DESC * GetUnconvertedRootSignatureDesc();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12.h
Library	D3d12.lib
DLL	D3d12.dll

See also

[ID3D12VersionedRootSignatureDeserializer](#)

[Root Signature Version 1.1](#)

d3d12sdklayers.h header

4/29/2020 • 2 minutes to read • [Edit Online](#)

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

- [Direct3D 12 Graphics](#) d3d12sdklayers.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
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.
ID3D12Debug3	Adds configurable levels of GPU-based validation to the debug layer.
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.
ID3D12SharingContract	Part of a contract between D3D11On12 diagnostic layers and graphics diagnostics tools.

Structures

TITLE	DESCRIPTION
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.

TITLE	DESCRIPTION
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_DESC	Allow or deny certain types of messages to pass through a filter.
D3D12_MESSAGE	A debug message in the Information Queue.

Enumerations

TITLE	DESCRIPTION
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_RLDO_FLAGS	Specifies options for the amount of information to report about a live device object's lifetime.

D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12sdklayers.h (include D3d12sdklayers_RS1.h)

See also

[Debug Layer Structures](#)

[SetEnableGPUBasedValidation](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates the debug parameter type used by [ID3D12DebugCommandList1::SetDebugParameter](#) and [ID3D12DebugCommandList1::GetDebugParameter](#).

Syntax

```
typedef enum D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE {
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_GPU_BASED_VALIDATION_SETTINGS
} ;
```

Constants

D3D12_DEBUG_COMMAND_LIST_PARAMETER_GPU_BASED_VALIDATION_SETTINGS	Indicates the parameter is type D3D12_DEBUG_COMMAND_LIST_GPU_BASED_VALIDATION_SETTINGS .
--	--

Requirements

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes settings used by GPU-Based Validation.

Syntax

```
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

Header	d3d12sdklayers.h (include D3d12sdklayers_RS1.h)

See also

[Debug Layer Structures](#)

[SetEnableGPUBasedValidation](#)

D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12sdklayers.h (include D3D12.h)
--------	------------------------------------

See also

[Core Structures](#)

D3D12_DEBUG_DEVICE_PARAMETER_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the data type of the memory pointed to by the *pData* parameter of [ID3D12DebugDevice1::SetDebugParameter](#) and [ID3D12DebugDevice1::GetDebugParameter](#).

Syntax

```
typedef enum D3D12_DEBUG_DEVICE_PARAMETER_TYPE {
    D3D12_DEBUG_DEVICE_PARAMETER_FEATURE_FLAGS,
    D3D12_DEBUG_DEVICE_PARAMETER_GPU_BASED_VALIDATION_SETTINGS,
    D3D12_DEBUG_DEVICE_PARAMETER_GPU_SLOWDOWN_PERFORMANCE_FACTOR
} ;
```

Constants

D3D12_DEBUG_DEVICE_PARAMETER_FEATURE_FLAGS	Indicates <i>pData</i> points to a D3D12_DEBUG_FEATURE value.
D3D12_DEBUG_DEVICE_PARAMETER_GPU_BASED_VALIDATION_SETTINGS	Indicates <i>pData</i> points to a D3D12_DEBUG_DEVICE_GPU_BASED_VALIDATION_SETTINGS structure.
D3D12_DEBUG_DEVICE_PARAMETER_GPU_SLOWDOWN_PERFORMANCE_FACTOR	Indicates <i>pData</i> points to a D3D12_DEBUG_DEVICE_GPU_SLOWDOWN_PERFORMANCE_FACTOR structure.

Requirements

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

D3D12_DEBUG_FEATURE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Flags for optional D3D12 Debug Layer features.

Syntax

```
typedef enum D3D12_DEBUG_FEATURE {
    D3D12_DEBUG_FEATURE_NONE,
    D3D12_DEBUG_FEATURE_ALLOW_BEHAVIOR_CHANGING_DEBUG_AIDS,
    D3D12_DEBUG_FEATURE_CONSERVATIVE_RESOURCE_STATE_TRACKING,
    D3D12_DEBUG_FEATURE_DISABLE_VIRTUALIZED_BUNDLES_VALIDATION,
    D3D12_DEBUG_FEATURE_EMULATE_WINDOWS7
} ;
```

Constants

D3D12_DEBUG_FEATURE_NONE	The default. No optional Debug Layer features.
D3D12_DEBUG_FEATURE_ALLOW_BEHAVIOR_CHANGING_DEBUG_AIDS	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	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	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

Header	d3d12sdklayers.h
--------	------------------

See also

Debug Layer Enumerations

D3D12_GPU_BASED_VALIDATION_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes the level of GPU-based validation to perform at runtime.

Syntax

```
typedef enum D3D12_GPU_BASED_VALIDATION_FLAGS {  
    D3D12_GPU_BASED_VALIDATION_FLAGS_NONE,  
    D3D12_GPU_BASED_VALIDATION_FLAGS_DISABLE_STATE_TRACKING  
} ;
```

Constants

D3D12_GPU_BASED_VALIDATION_FLAGS_NONE	Default behavior; resource states, descriptors, and descriptor tables are all validated.
D3D12_GPU_BASED_VALIDATION_FLAGS_DISABLE_STATE_TRACKING	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::SetGPUBasedValidationFlags](#) method to configure the amount of runtime validation that will occur.

Requirements

Header	d3d12sdklayers.h
--------	------------------

See also

[Core Enumerations](#)

D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies how GPU-Based Validation handles patched pipeline states during [ID3D12Device::CreateGraphicsPipelineState](#) and [ID3D12Device::CreateComputePipelineState](#).

Syntax

```
typedef enum D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS {  
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_NONE,  
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_TRACKING_ONLY_SHADERS,  
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_UNGUARDED_VALIDATION_SHADERS,  
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_GUARDED_VALIDATION_SHADERS,  
    D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS_VALID_MASK  
} ;
```

Constants

D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_NONE	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.
D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_TRACKING_ONLY_SHADERS	Indicates that state-tracking GPU-Based Validation PSO's should be created along with the original PSO at create time.
D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_UNGUARDED_VALIDATION_SHADERS	Indicates that unguarded GPU-Based Validation PSO's should be created along with the original PSO at create time.
D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAG_FRONT_LOAD_CREATE_GUARDED_VALIDATION_SHADERS	Indicates that guarded GPU-Based Validation PSO's should be created along with the original PSO at create time.
D3D12_GPU_BASED_VALIDATION_PIPELINE_STATE_CREATE_FLAGS_VALID_MASK	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

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the type of shader patching used by GPU-Based Validation at either the device or command list level.

Syntax

```
typedef enum D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE {  
    D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_NONE,  
    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

D3D12_GPU_BASED_VALIDATION_SHADER_PATCH_MODE_NONE	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

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

D3D12_INFO_QUEUE_FILTER structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Header	d3d12sdklayers.h

See also

[Debug Layer Structures](#)

D3D12_INFO_QUEUE_FILTER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

`pIDList`

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

`Header`

d3d12sdklayers.h

See also

[Debug Layer Structures](#)

D3D12_MESSAGE structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A debug message in the Information Queue.

Syntax

```
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

Header	d3d12sdklayers.h

See also

[Debug Layer Structures](#)

D3D12_MESSAGE_CATEGORY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_MESSAGE_CATEGORY {
    D3D12_MESSAGE_CATEGORY_APPLICATION_DEFINED,
    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

D3D12_MESSAGE_CATEGORY_APPLICATION_DEFINED	Indicates a user defined message, see ID3D12InfoQueue::AddMessage .
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	

Remarks

This is part of the Information Queue feature, refer to the [ID3D12InfoQueue](#) Interface.

Requirements

Header	
	d3d12sdklayers.h

See also

[Debug Layer Enumerations](#)

D3D12_MESSAGE_ID enumeration

5/27/2020 • 8 minutes to read • [Edit Online](#)

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

```
typedef enum D3D12_MESSAGE_ID {  
    D3D12_MESSAGE_ID_UNKNOWN,  
    D3D12_MESSAGE_ID_STRING_FROM_APPLICATION,  
    D3D12_MESSAGE_ID_CORRUPTED_THIS,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER1,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER2,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER3,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER4,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETERS,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER6,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER7,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER8,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER9,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER10,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER11,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER12,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER13,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER14,  
    D3D12_MESSAGE_ID_CORRUPTED_PARAMETER15,  
    D3D12_MESSAGE_ID_CORRUPTED_MULTITHREADING,  
    D3D12_MESSAGE_ID_MESSAGE_REPORTING_OUTOFMEMORY,  
    D3D12_MESSAGE_ID_GETPRIVATEDATA_MOREDATA,  
    D3D12_MESSAGE_ID_SETPRIVATEDATA_INVALIDFREEDATA,  
    D3D12_MESSAGE_ID_SETPRIVATEDATA_CHANGINGPARAMS,  
    D3D12_MESSAGE_ID_SETPRIVATEDATA_OUTOFMEMORY,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_UNRECOGNIZEDFORMAT,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDESC,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFORMAT,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDVIDEOPLANESLICE,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDPLANESLICE,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDIMENSIONS,  
    D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDRESOURCE,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNRECOGNIZEDFORMAT,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNSUPPORTEDFORMAT,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDESC,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDFORMAT,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDVIDEOPLANESLICE,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDPLANESLICE,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDIMENSIONS,  
    D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDRESOURCE,  
    D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_UNRECOGNIZEDFORMAT,  
    D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDESC,  
    D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFORMAT,  
    D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDDIMENSIONS,  
    D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDRESOURCE,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_OUTOFMEMORY,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TOOMANYELEMENTS,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDFORMAT,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INCOMPATIBLEFORMAT,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOT,  
    D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDINPUTSLOTCLASS,  
    D3D12_MESSAGE_ID_CREATETNPLTI_AVAIL_STDRATESIOTCI_ASMSMTSMATCH  
};
```

```
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_STEPRATESELECTCLASSMISMATCH,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSLOTCLASSCHANGE,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSTEPRATECHANGE,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDALIGNMENT,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_DUPLICATESEMANTIC,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_UNPARSEABLEINPUTSIGNATURE,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_NULLSEMANTIC,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_MISSINGELEMENT,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_OUTOFCMEMORY,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_OUTOFCMEMORY,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTOFCMEMORY,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMENTRIES,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSTREAMSTRIDEUNUSED,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSLOT0EXPECTED,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSLOT,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_ONLYONEELEMENTPERSLOT,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCOMPONENTCOUNT,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTARTCOMPONENTANDCOMPONENTCOUNT,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDGAPDEFINITION,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_REPEATEOUPUT,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSTREAMSTRIDE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGSEMANTIC,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MASKMISMATCH,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_CANTHAVEONLYGAPS,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DECLTOOCOMPLEX,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGOUTPUTSIGNATURE,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_OUTOFCMEMORY,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFILLMODE,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDCULLMODE,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDDEPTHBIASCLAMP,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDSLOPESCALEDEDPHTBIAS,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHWRITEMASK,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDDEPTHFUNC,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACESTENCILFAIL0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACESTENCILZFAIL0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACESTENCILPASS0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFRONTFACESTENCILFUNC,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACESTENCILFAIL0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACESTENCILZFAIL0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACESTENCILPASS0P,
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDBACKFACESTENCILFUNC,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEND,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLENDALPHA,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLENDALPHA,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOPALPHA,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDRENDERTARGETWRITEMASK,
D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_INVALID,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_STRIDE_TOO_SMALL,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_TOO_SMALL,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_FORMAT_INVALID,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_TOO_SMALL,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INVALID_PRIMITIVETOPOLOGY,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_STRIDE_UNALIGNED,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_OFFSET_UNALIGNED,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_AT_FAULT,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_DODGYLY_ATFAULT
```

```
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_POSSIBLY_AT_FAULT,
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_NOT_AT_FAULT,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TRAILING_DIGIT_IN_SEMANTIC,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_TRAILING_DIGIT_IN_SEMANTIC,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TYPE_MISMATCH,
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_EMPTY_LAYOUT,
D3D12_MESSAGE_ID_LIVE_OBJECT_SUMMARY,
D3D12_MESSAGE_ID_LIVE_DEVICE,
D3D12_MESSAGE_ID_LIVE_SWAPCHAIN,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILVIEW_INVALIDFLAGS,
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAMTORASTERIZER,
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTREAM,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDENTRIES,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UNEXPECTEDSTRIDES,
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMSTRIDES,
D3D12_MESSAGE_ID_CREATEHULLSHADER_OUTOFGMEMORY,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_OUTOFGMEMORY,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_RESOURCE_UNMAP_NOTMAPPED,
D3D12_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_MISMATCHED_DATA_SIZE,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_OUTOFGMEMORY,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCLASSLINKAGE,
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEFLOATOPSNOTSUPPORTED,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDRESOURCE,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDESC,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFORMAT,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDVIDEOPLANESLICE,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDPLANESLICE,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDIMENSIONS,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_UNRECOGNIZEDFORMAT,
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFLAGS,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFORCEDSAMPLECOUNT,
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDLOGICOPS,
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEEXTENSIONSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_UAVSNOTSUPPORTED,
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_INVALIDSOURCERECT,
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_EMPTYRECT,
D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_INVALID_PARAMETER,
D3D12_MESSAGE_ID_COPYTILEMAPPINGS_INVALID_PARAMETER,
D3D12_MESSAGE_ID_CREATEDEVICE_INVALIDARGS,
D3D12_MESSAGE_ID_CREATEDEVICE_WARNING,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_TYPE,
```

```
D3D12_MESSAGE_ID_RESOURCE_BARRIER_NULL_POINTER,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SUBRESOURCE,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_RESERVED_BITS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISSING_BIND_FLAGS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_MISC_FLAGS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MATCHING_STATES,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINATION,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_BEFORE_AFTER_MISMATCH,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_RESOURCE,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_SAMPLE_COUNT,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINED_FLAGS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS_FOR_FORMAT,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SPLIT_BARRIER,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_END,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_BEGIN,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAG,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMMAND_LIST_TYPE,
D3D12_MESSAGE_ID_INVALID_SUBRESOURCE_STATE,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CONTENTION,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET_BUNDLE,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CANNOT_RESET,
D3D12_MESSAGE_ID_COMMAND_LIST_OPEN,
D3D12_MESSAGE_ID_INVALID_BUNDLE_API,
D3D12_MESSAGE_ID_COMMAND_LIST_CLOSED,
D3D12_MESSAGE_ID_WRONG_COMMAND_ALLOCATOR_TYPE,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_SYNC,
D3D12_MESSAGE_ID_COMMAND_LIST_SYNC,
D3D12_MESSAGE_ID_SET_DESCRIPTOR_HEAP_INVALID,
D3D12_MESSAGE_ID_CREATE_COMMANDQUEUE,
D3D12_MESSAGE_ID_CREATE_COMMANDALLOCATOR,
D3D12_MESSAGE_ID_CREATE_PIPELINESTATE,
D3D12_MESSAGE_ID_CREATE_COMMANDLIST12,
D3D12_MESSAGE_ID_CREATE_RESOURCE,
D3D12_MESSAGE_ID_CREATE_DESCRIPTORHEAP,
D3D12_MESSAGE_ID_CREATE_ROOTSIGNATURE,
D3D12_MESSAGE_ID_CREATE_LIBRARY,
D3D12_MESSAGE_ID_CREATE_HEAP,
D3D12_MESSAGE_ID_CREATE_MONITOREDFENCE,
D3D12_MESSAGE_ID_CREATE_QUERYHEAP,
D3D12_MESSAGE_ID_CREATE_COMMANDSIGNATURE,
D3D12_MESSAGE_ID_LIVE_COMMANDQUEUE,
D3D12_MESSAGE_ID_LIVE_COMMANDALLOCATOR,
D3D12_MESSAGE_ID_LIVE_PIPELINESTATE,
D3D12_MESSAGE_ID_LIVE_COMMANDLIST12,
D3D12_MESSAGE_ID_LIVE_RESOURCE,
D3D12_MESSAGE_ID_LIVE_DESCRIPTORHEAP,
D3D12_MESSAGE_ID_LIVE_ROOTSIGNATURE,
D3D12_MESSAGE_ID_LIVE_LIBRARY,
D3D12_MESSAGE_ID_LIVE_HEAP,
D3D12_MESSAGE_ID_LIVE_MONITOREDFENCE,
D3D12_MESSAGE_ID_LIVE_QUERYHEAP,
D3D12_MESSAGE_ID_LIVE_COMMANDSIGNATURE,
D3D12_MESSAGE_ID_DESTROY_COMMANDQUEUE,
D3D12_MESSAGE_ID_DESTROY_COMMANDALLOCATOR,
D3D12_MESSAGE_ID_DESTROY_PIPELINESTATE,
D3D12_MESSAGE_ID_DESTROY_COMMANDLIST12,
D3D12_MESSAGE_ID_DESTROY_RESOURCE,
D3D12_MESSAGE_ID_DESTROY_DESCRIPTORHEAP,
D3D12_MESSAGE_ID_DESTROY_ROOTSIGNATURE,
D3D12_MESSAGE_ID_DESTROY_LIBRARY,
D3D12_MESSAGE_ID_DESTROY_HEAP,
D3D12_MESSAGE_ID_DESTROY_MONITOREDFENCE,
D3D12_MESSAGE_ID_DESTROY_QUERYHEAP,
D3D12_MESSAGE_ID_DESTROY_COMMANDSIGNATURE,
D3D12_MESSAGE_ID_CREATEROBJECT_INVALIDDIMENSIONS,
D3D12_MESSAGE_ID_CREATEROBJECT_INVALIDMISCFLAGS,
D3D12_MESSAGE_ID_CREATEROBJECT_INVALIDARG_RETURN,
```

```
D3D12_MESSAGE_ID_CREATERESOURCE_OUTOFMEMORY_RETURN,
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDESC,
D3D12_MESSAGE_ID_POSSIBLY_INVALID_SUBRESOURCE_STATE,
D3D12_MESSAGE_ID_INVALID_USE_OF_NON_RESIDENT_RESOURCE,
D3D12_MESSAGE_ID_POSSIBLE_INVALID_USE_OF_NON_RESIDENT_RESOURCE,
D3D12_MESSAGE_ID_BUNDLE_PIPELINE_STATE_MISMATCH,
D3D12_MESSAGE_ID_PRIMITIVE_TOPOLOGY_MISMATCH_PIPELINE_STATE,
D3D12_MESSAGE_ID_RENDER_TARGET_FORMAT_MISMATCH_PIPELINE_STATE,
D3D12_MESSAGE_ID_RENDER_TARGET_SAMPLE_DESC_MISMATCH_PIPELINE_STATE,
D3D12_MESSAGE_ID_DEPTH_STENCIL_FORMAT_MISMATCH_PIPELINE_STATE,
D3D12_MESSAGE_ID_DEPTH_STENCIL_SAMPLE_DESC_MISMATCH_PIPELINE_STATE,
D3D12_MESSAGE_ID_CREATESHADER_INVALIDBYTECODE,
D3D12_MESSAGE_ID_CREATEHEAP_NULLDESC,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDSIZE,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDHEATYPE,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMEMORYPOOL,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDPROPERTIES,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDALIGNMENT,
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMISCFLAGS,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDMISCFLAGS,
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDARG_RETURN,
D3D12_MESSAGE_ID_CREATEHEAP_OUTOFMEMORY_RETURN,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLHEAPPROPERTIES,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEATYPE,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDMEMORYPOOL,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPPROPERTIES,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPMISCFLAGS,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPMISCFLAGS,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDARG_RETURN,
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_OUTOFMEMORY_RETURN,
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_UNRECOGNIZEDHEATYPE,
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_INVALIDHEATYPE,
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_INVALID_DESC,
D3D12_MESSAGE_ID_INVALID_DESCRIPTOR_HANDLE,
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALID_CONSERVATIVERASTERMODE,
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_RESOURCE,
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_DESC,
D3D12_MESSAGE_ID_CREATE_UNORDEREDACCESS_VIEW_INVALID_COUNTER_USAGE,
D3D12_MESSAGE_ID_COPY_DESCRIPTORSS_INVALID_RANGES,
D3D12_MESSAGE_ID_COPY_DESCRIPTORSS_WRITE_ONLY_DESCRIPTOR,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_RT_V_FORMAT_NOT_UNKNOWN,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_RENDER_TARGET_COUNT,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_VERTEX_SHADER_NOT_SET,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INPUTLAYOUT_NOT_SET,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_HS_DS_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_REGISTERINDEX,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_COMPONENTTYPE,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_REGISTERMASK,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_SYSTEMVALUE,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_NEVERWRITTEN_ALWAYSREADS,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_MINPRECISION,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_SEMANTICNAME_NOT_FOUND,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_XOR_DS_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HULL_SHADER_INPUT_TOPOLOGY_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_DS_CONTROL_POINT_COUNT_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_HS_DS_TESSELLATOR_DOMAIN_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_USE_OF_CENTER_MULTISAMPLE_PATTERN,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_USE_OF_FORCED_SAMPLE_COUNT,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_PRIMITIVETOPOLOGY,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_SYSTEMVALUE,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_OM_DUAL_SOURCE_BLENDING_CAN_ONLY_HAVE_RENDER_TARGET_0,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_OM_RENDER_TARGET_DOES_NOT_SUPPORT_BLENDING,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_PS_OUTPUT_TYPE_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_OM_RENDER_TARGET_DOES_NOT_SUPPORT_LOGIC_OPS,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_RENDERTARGETVIEW_NOT_SET,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_DEPTHSTENCILVIEW_NOT_SET,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_GS_INPUT_PRIMITIVE_MISMATCH,
```

```
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Position_Not_Present,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Missing_Root_Signature_Flags,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Index_Buffer_Properties,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Sample_Desc,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_HS_Root_Signature_Mismatch,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_DS_Root_Signature_Mismatch,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_VS_Root_Signature_Mismatch,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_GS_Root_Signature_Mismatch,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_PS_Root_Signature_Mismatch,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Missing_Root_Signature,
D3D12_MESSAGE_ID_Execute_Bundle_Open_Bundle,
D3D12_MESSAGE_ID_Execute_Bundle_Descriptor_Heap_Mismatch,
D3D12_MESSAGE_ID_Execute_Bundle_Type,
D3D12_MESSAGE_ID_Draw_Empty_Scissor_Rectangle,
D3D12_MESSAGE_ID_Create_Root_Signature_Blob_Not_Found,
D3D12_MESSAGE_ID_Create_Root_Signature_Deserialize_Failed,
D3D12_MESSAGE_ID_Create_Root_Signature_Invalid_Configuration,
D3D12_MESSAGE_ID_Create_Root_Signature_Not_Supported_On_Device,
D3D12_MESSAGE_ID_Createresourceandheap_NullResourceProperties,
D3D12_MESSAGE_ID_Createresourceandheap_NullHeap,
D3D12_MESSAGE_ID_GetResourceAllocationInfo_InvalidRdescs,
D3D12_MESSAGE_ID_Makeresident_Nullobjectarray,
D3D12_MESSAGE_ID_Evict_Nullobjectarray,
D3D12_MESSAGE_ID_Set_Descriptor_Table_Invalid,
D3D12_MESSAGE_ID_Set_Root_Constant_Invalid,
D3D12_MESSAGE_ID_Set_Root_Constant_Buffer_View_Invalid,
D3D12_MESSAGE_ID_Set_Root_Shader_Resource_View_Invalid,
D3D12_MESSAGE_ID_Set_Root_Unordered_Access_View_Invalid,
D3D12_MESSAGE_ID_Set_Vertex_Buffers_Invalid_Desc,
D3D12_MESSAGE_ID_Set_Index_Buffer_Invalid_Desc,
D3D12_MESSAGE_ID_Set_Stream_Output_Buffers_Invalid_Desc,
D3D12_MESSAGE_ID_Createresource_UnrecognizedDimensionality,
D3D12_MESSAGE_ID_Createresource_UnrecognizedLayout,
D3D12_MESSAGE_ID_Createresource_InvalidDimensionality,
D3D12_MESSAGE_ID_Createresource_InvalidAlignment,
D3D12_MESSAGE_ID_Createresource_InvalidMiplevels,
D3D12_MESSAGE_ID_Createresource_Invalidsampledesc,
D3D12_MESSAGE_ID_Createresource_InvalidLayout,
D3D12_MESSAGE_ID_Set_Index_Buffer_Invalid,
D3D12_MESSAGE_ID_Set_Vertex_Buffers_Invalid,
D3D12_MESSAGE_ID_Set_Stream_Output_Buffers_Invalid,
D3D12_MESSAGE_ID_Set_Render_Targets_Invalid,
D3D12_MESSAGE_ID_CreateQuery_Heap_Invalid_Parameters,
D3D12_MESSAGE_ID_Begin_End_Query_Invalid_Parameters,
D3D12_MESSAGE_ID_Close_Command_List_Open_Query,
D3D12_MESSAGE_ID_Resolve_Query_Data_Invalid_Parameters,
D3D12_MESSAGE_ID_Set_Predication_Invalid_Parameters,
D3D12_MESSAGE_ID_Timestamps_Not_Supported,
D3D12_MESSAGE_ID_Createresource_UnrecognizedFormat,
D3D12_MESSAGE_ID_Createresource_InvalidFormat,
D3D12_MESSAGE_ID_GetCopyableFootprints_InvalidSubresourceRange,
D3D12_MESSAGE_ID_GetCopyableFootprints_InvalidBaseOffset,
D3D12_MESSAGE_ID_GetCopyableLayout_InvalidSubresourceRange,
D3D12_MESSAGE_ID_GetCopyableLayout_InvalidBaseOffset,
D3D12_MESSAGE_ID_Resource_BARRIER_Invalid_Heap,
D3D12_MESSAGE_ID_Create_Sampler_Invalid,
D3D12_MESSAGE_ID_CreateCommandSignature_Invalid,
D3D12_MESSAGE_ID_Execute_Indirect_Invalid_Parameters,
D3D12_MESSAGE_ID_GetGPUVirtualAddress_Invalid_Resource_Dimension,
D3D12_MESSAGE_ID_Createresource_InvalidClearValue,
D3D12_MESSAGE_ID_Createresource_UnrecognizedClearValueFormat,
D3D12_MESSAGE_ID_Createresource_InvalidClearValueFormat,
D3D12_MESSAGE_ID_Createresource_ClearValueDenormFlush,
D3D12_MESSAGE_ID_ClearRenderTargetView_MismatchingClearValue,
D3D12_MESSAGE_ID_ClearDepthStencilView_MismatchingClearValue,
D3D12_MESSAGE_ID_Map_InvalidHeap,
D3D12_MESSAGE_ID_Unmap_InvalidHeap,
D3D12_MESSAGE_ID_Map_InvalidResource,
D3D12_MESSAGE_ID_Unmap_InvalidResource,
```

```
- - - - -
D3D12_MESSAGE_ID_MAP_INVALIDSUBRESOURCE,
D3D12_MESSAGE_ID_UNMAP_INVALIDSUBRESOURCE,
D3D12_MESSAGE_ID_MAP_INVALIDRANGE,
D3D12_MESSAGE_ID_UNMAP_INVALIDRANGE,
D3D12_MESSAGE_ID_MAP_INVALIDDATAPORTER,
D3D12_MESSAGE_ID_MAP_INVALIDARG_RETURN,
D3D12_MESSAGE_ID_MAP_OUTOFPMEMORY_RETURN,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_BUNDLENOTSUPPORTED,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_COMMANDLISTMISMATCH,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_OPENCOMMANDLIST,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_FAILEDCOMMANDLIST,
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLDST,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDDDSTRESOURCEDIMENSION,
D3D12_MESSAGE_ID_COPYBUFFERREGION_DSTRANGEOUTOFCOMMANDLIST,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDSRCRESOURCEDIMENSION,
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLSRC,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDDDSTRESOURCEDIMENSION,
D3D12_MESSAGE_ID_COPYBUFFERREGION_SRCRANGEOUTOFCOMMANDLIST,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDCOPYFLAGS,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLDST,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDDSTTYPE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTRESOURCEDIMENSION,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTRESOURCE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTSUBRESOURCE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTOFFSET,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDDDSTFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTDIMENSIONS,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTROWPITCH,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTPLACEMENT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTDPLACEDFOOTPRINTFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_DSTREGIONOUTOFCOMMANDLIST,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLSRC,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCTYPE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCEDIMENSION,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRESOURCE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCSUBRESOURCE,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCOFFSET,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZEDSRCFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDIMENSIONS,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCROWPITCH,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCPLACEMENT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDSPLACEDFOOTPRINTFORMAT,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_SRCREGIONOUTOFCOMMANDLIST,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDDSTCOORDINATES,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCBOX,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_FORMATMISMATCH,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_EMPTYBOX,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDCOPYFLAGS,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SUBRESOURCE_INDEX,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_FORMAT,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_MISMATCH,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SAMPLE_COUNT,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_INVALID_SHADER,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_CS_ROOT_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_MISSING_ROOT_SIGNATURE,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALIDCACHEDBLOB,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBADAPTERMISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDRIVERVERSIONMISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDESCMISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBIGNORED,
D3D12_MESSAGE_ID_Writetosubresource_INVALIDHEAP,
D3D12_MESSAGE_ID_Writetosubresource_INVALIDRESOURCE,
D3D12_MESSAGE_ID_Writetosubresource_INVALIDBOX,
D3D12_MESSAGE_ID_Writetosubresource_INVALIDSUBRESOURCE,
D3D12_MESSAGE_ID_Writetosubresource_EMPTYBOX,
D3D12_MESSAGE_ID_Readfromsubresource_INVALIDHEAP,
D3D12_MESSAGE_ID_Readfromsubresource_INVALIDRESOURCE,
D3D12_MESSAGE_ID_Readfromsubresource_INVALIDTDRX.
```

```
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDSUBRESOURCE,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDSUBRESOURCE,
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_EMPTYBOX,
D3D12_MESSAGE_ID_TOO_MANY_NODES_SPECIFIED,
D3D12_MESSAGE_ID_INVALID_NODE_INDEX,
D3D12_MESSAGE_ID_GETTHEAPPROPERTIES_INVALIDRESOURCE,
D3D12_MESSAGE_ID_NODE_MASK_MISMATCH,
D3D12_MESSAGE_ID_COMMAND_LIST_OUTOFMEMORY,
D3D12_MESSAGE_ID_COMMAND_LIST_MULTIPLE_SWAPCHAIN_BUFFER_REFERENCES,
D3D12_MESSAGE_ID_COMMAND_LIST_TOO_MANY_SWAPCHAIN_REFERENCES,
D3D12_MESSAGE_ID_COMMAND_QUEUE_TOO_MANY_SWAPCHAIN_REFERENCES,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_WRONGSWAPCHAINBUFFERREFERENCE,
D3D12_MESSAGE_ID_COMMAND_LIST_SETRENDERTARGETS_INVALIDNUMRENDERTARGETS,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_TYPE,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_FLAGS,
D3D12_MESSAGE_ID_CREATESHAREDRSOURCE_INVALIDFLAGS,
D3D12_MESSAGE_ID_CREATESHAREDRSOURCE_INVALIDFORMAT,
D3D12_MESSAGE_ID_CREATESHAREDHEAP_INVALIDFLAGS,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_UNRECOGNIZEDPROPERTIES,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDSIZE,
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDOBJECT,
D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDOBJECT,
D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDKEY,
D3D12_MESSAGE_ID_KEYEDMUTEX_WRONGSTATE,
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_PRIORITY,
D3D12_MESSAGE_ID_OBJECT_DELETED_WHILE_STILL_IN_USE,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_FLAGS,
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_HAS_NO_RESOURCE,
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_RENDER_TARGET_DELETED,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_ALL_RENDER_TARGETS_HAVE_UNKNOWN_FORMAT,
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_INTERSECTS_MULTIPLE_BUFFERS,
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_GPU_WRITTEN_READBACK_RESOURCE_MAPPED,
D3D12_MESSAGE_ID_UNMAP_RANGE_NOT_EMPTY,
D3D12_MESSAGE_ID_MAP_INVALID_NULLRANGE,
D3D12_MESSAGE_ID_UNMAP_INVALID_NULLRANGE,
D3D12_MESSAGE_ID_NO_GRAPHICS_API_SUPPORT,
D3D12_MESSAGE_ID_NO_COMPUTE_API_SUPPORT,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_FLAGS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_ARGUMENT_UNINITIALIZED,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_HEAP_INDEX_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TABLE_REGISTER_INDEX_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_UNINITIALIZED,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TYPE_MISMATCH,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SRV_RESOURCE_DIMENSION_MISMATCH,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UAV_RESOURCE_DIMENSION_MISMATCH,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INCOMPATIBLE_RESOURCE_STATE,
D3D12_MESSAGE_ID_COPYRESOURCE_NULLDST,
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDDSTRESOURCE,
D3D12_MESSAGE_ID_COPYRESOURCE_NULLSRC,
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDSRCRESOURCE,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLDST,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDDSTRESOURCE,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLSRC,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDSRCRESOURCE,
D3D12_MESSAGE_ID_PIPELINE_STATE_TYPE_MISMATCH,
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_ZERO_BARRIERS,
D3D12_MESSAGE_ID_BEGIN_END_EVENT_MISMATCH,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_POSSIBLE_BEFORE_AFTER_MISMATCH,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_BEGIN_END,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INVALID_RESOURCE,
D3D12_MESSAGE_ID_USE_OF_ZERO_REFCOUNT_OBJECT,
D3D12_MESSAGE_ID_OBJECT_EVICTED_WHILE_STILL_IN_USE,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_DESCRIPTOR_ACCESS_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_INVALIDLIBRARYBLOB,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_DRIVERVERSIONMISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_ADAPTERVERSIONMISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATE_PIPELINELIBRARY_TDRADY
```

```
D3D12_MESSAGE_ID_CREATE_PIPELINELIBRARY,
D3D12_MESSAGE_ID_LIVE_PIPELINELIBRARY,
D3D12_MESSAGE_ID_DESTROY_PIPELINELIBRARY,
D3D12_MESSAGE_ID_STOREPIPELINE_NONAME,
D3D12_MESSAGE_ID_STOREPIPELINE_DUPLICATENAME,
D3D12_MESSAGE_ID_LOADPIPELINE_NAMENOTFOUND,
D3D12_MESSAGE_ID_LOADPIPELINE_INVALIDDESC,
D3D12_MESSAGE_ID_PIPELINELIBRARY_SERIALIZE_NOTENOUGHMEMORY,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_PS_OUTPUT_RT_OUTPUT_MISMATCH,
D3D12_MESSAGE_ID_SETEVENTONMULTIPLEFENCECOMPLETION_INVALIDFLAGS,
D3D12_MESSAGE_ID_CREATE_QUEUE_VIDEO_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATE_COMMAND_ALLOCATOR_VIDEO_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_VIDEO_DECODE_STATISTICS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDLIST,
D3D12_MESSAGE_ID_CREATE_VIDEODECODER,
D3D12_MESSAGE_ID_CREATE_VIDEODECODESTREAM,
D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDLIST,
D3D12_MESSAGE_ID_LIVE_VIDEODECODER,
D3D12_MESSAGE_ID_LIVE_VIDEODECODESTREAM,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDLIST,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODER,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODESTREAM,
D3D12_MESSAGE_ID_DECODER_FRAME_INVALID_PARAMETERS,
D3D12_MESSAGE_ID_DEPRECATED_API,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_COMMAND_LIST_TYPE,
D3D12_MESSAGE_ID_COMMAND_LIST_DESCRIPTOR_TABLE_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_CONSTANT_BUFFER_VIEW_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_SHADER_RESOURCE_VIEW_NOT_SET,
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_UNORDERED_ACCESS_VIEW_NOT_SET,
D3D12_MESSAGE_ID_DISCARD_INVALID_SUBRESOURCE_RANGE,
D3D12_MESSAGE_ID_DISCARD_ONE_SUBRESOURCE_FOR_MIPS_WITH_RECTS,
D3D12_MESSAGE_ID_DISCARD_NO_RECTS_FOR_NON_TEXTURE2D,
D3D12_MESSAGE_ID_COPY_ON_SAME_SUBRESOURCE,
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PAGEABLE,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UNSUPPORTED,
D3D12_MESSAGE_ID_STATIC_DESCRIPTOR_INVALID_DESCRIPTOR_CHANGE,
D3D12_MESSAGE_ID_DATA_STATIC_DESCRIPTOR_INVALID_DATA_CHANGE,
D3D12_MESSAGE_ID_DATA_STATIC_WHILE_SET_AT_EXECUTE_DESCRIPTOR_INVALID_DATA_CHANGE,
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_STATIC_DESCRIPTOR_DATA_STATIC_NOT_SET,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_ACCESS_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SAMPLER_MODE_MISMATCH,
D3D12_MESSAGE_ID_CREATE_FENCE_INVALID_FLAGS,
D3D12_MESSAGE_ID_RESOURCE_BARRIER_DUPLICATE_SUBRESOURCE_TRANSITIONS,
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PRIORITY,
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_LARGE_NUM_DESCRIPTORS,
D3D12_MESSAGE_ID_BEGIN_EVENT,
D3D12_MESSAGE_ID_END_EVENT,
D3D12_MESSAGE_ID_CREATEDevice_DEBUG_LAYER_STARTUP_OPTIONS,
D3D12_MESSAGE_ID_CREATEDEPTH_STENCILSTATE_DEPTHBOUNDSTEST_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_DUPLICATE_SUBOBJECT,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_UNKNOWN_SUBOBJECT,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_ZERO_SIZE_STREAM,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_STREAM,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CANNOT_DEDUCE_TYPE,
D3D12_MESSAGE_ID_COMMAND_LIST_STATIC_DESCRIPTOR_RESOURCE_DIMENSION_MISMATCH,
D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_PRIVILEGE_FOR_GLOBAL_REALTIME,
D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_HARDWARE_SUPPORT_FOR_GLOBAL_REALTIME,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_ARCHITECTURE,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DST,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE_DIMENSION,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DST_RANGE_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_SRC,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE_DIMENSION,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_SRC_RANGE_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_OFFSET_ALIGNMENT,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_RESOURCES,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_SUBRESOURCE_RANGES,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_RESOURCE,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_SUBRESOURCE_RANGE,
```

```
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_SUBRESOURCE_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_RANGE_OUT_OF_BOUNDS,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_ZERO_DEPENDENCIES,
D3D12_MESSAGE_ID_DEVICE_CREATE_SHARED_HANDLE_INVALIDARG,
D3D12_MESSAGE_ID_DESCRIPTOR_HANDLE_WITH_INVALID_RESOURCE,
D3D12_MESSAGE_ID_SETDEPTHBOUNDS_INVALIDARGS,
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_STATE_IMPRECISE,
D3D12_MESSAGE_ID_COMMAND_LIST_PIPELINE_STATE_NOT_SET,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_SHADER_MODEL_MISMATCH,
D3D12_MESSAGE_ID_OBJECT_ACCESSED_WHILE_STILL_IN_USE,
D3D12_MESSAGE_ID_PROGRAMMABLE_MSAA_UNSUPPORTED,
D3D12_MESSAGE_ID_SETSAMPLEPOSITIONS_INVALIDARGS,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCEREGION_INVALID_RECT,
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDQUEUE,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDLIST,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDQUEUE,
D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDQUEUE,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDLIST,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDQUEUE,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDQUEUE,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDLIST,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDQUEUE,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSOR,
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSSTREAM,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSOR,
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSSTREAM,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSOR,
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSSTREAM,
D3D12_MESSAGE_ID_PROCESS_FRAME_INVALID_PARAMETERS,
D3D12_MESSAGE_ID_COPY_INVALIDLAYOUT,
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION,
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION_POLICY,
D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION,
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION,
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION_POLICY,
D3D12_MESSAGE_ID_LIVE_PROTECTED_RESOURCE_SESSION,
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION,
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION_POLICY,
D3D12_MESSAGE_ID_DESTROY_PROTECTED_RESOURCE_SESSION,
D3D12_MESSAGE_ID_PROTECTED_RESOURCE_SESSION_UNSUPPORTED,
D3D12_MESSAGE_ID_FENCE_INVALIDOPERATION,
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_COPY_QUEUE_TIMESTAMPS_NOT_SUPPORTED,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_DEFERRED,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMFIRSTUSE,
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMCLEAR,
D3D12_MESSAGE_ID_CREATE_VIDEODECODERHEAP,
D3D12_MESSAGE_ID_LIVE_VIDEODECODERHEAP,
D3D12_MESSAGE_ID_DESTROY_VIDEODECODERHEAP,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDARG_RETURN,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_OUTOFMEMORY_RETURN,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDADDRESS,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDHANDLE,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_DEST,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_MODE,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_ALIGNMENT,
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_NOT_SUPPORTED,
D3D12_MESSAGE_ID_SETVIEWINSTANCEMASK_INVALIDARGS,
D3D12_MESSAGE_ID_VIEW_INSTANCING_UNSUPPORTED,
D3D12_MESSAGE_ID_VIEW_INSTANCING_INVALIDARGS,
D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_DECODE_REFERENCE_ONLY_FLAG,
D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_DECODE_REFERENCE_ONLY_FLAG,
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_FAILURE,
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_UNSUPPORTED,
D3D12_MESSAGE_ID_VIDEO_decode_SUPPORT_INVALID_INPUT,
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODER_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_METADATA_ERROR,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_VIEW_INSTANCING_VERTEX_SIZE_EXCEEDED,
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_RUNTIME_INTERNAL_ERROR,
D3D12_MESSAGE_ID_NO_VIDEO_API_SUPPORT,
```

```
D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_INVALID_INPUT,
D3D12_MESSAGE_ID_CREATE_VIDEO_PROCESSOR_CAPS_FAILURE,
D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_UNSUPPORTED_FORMAT,
D3D12_MESSAGE_ID_VIDEO_DECODE_FRAME_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_ENQUEUE_MAKE_RESIDENT_INVALID_FLAGS,
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_UNSUPPORTED,
D3D12_MESSAGE_ID_VIDEO_PROCESS_FRAMES_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATE_COMMANDRECORDER,
D3D12_MESSAGE_ID_LIVE_COMMANDRECORDER,
D3D12_MESSAGE_ID_DESTROY_COMMANDRECORDER,
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_VIDEO_NOT_SUPPORTED,
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_SUPPORT_FLAGS,
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_FLAGS,
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_MORE_RECORDERS_THAN_LOGICAL_PROCESSORS,
D3D12_MESSAGE_ID_CREATE_COMMANDPOOL,
D3D12_MESSAGE_ID_LIVE_COMMANDPOOL,
D3D12_MESSAGE_ID_DESTROY_COMMANDPOOL,
D3D12_MESSAGE_ID_CREATE_COMMAND_POOL_INVALID_FLAGS,
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_VIDEO_NOT_SUPPORTED,
D3D12_MESSAGE_ID_COMMAND_RECORDER_SUPPORT_FLAGS_MISMATCH,
D3D12_MESSAGE_ID_COMMAND_RECORDER_CONTENTION,
D3D12_MESSAGE_ID_COMMAND_RECORDER_USAGE_WITH_CREATECOMMANDLIST_COMMAND_LIST,
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_USAGE_WITH_CREATECOMMANDLIST1_COMMAND_LIST,
D3D12_MESSAGE_ID_CANNOT_EXECUTE_EMPTY_COMMAND_LIST,
D3D12_MESSAGE_ID_CANNOT_RESET_COMMAND_POOL_WITH_OPEN_COMMAND_LISTS,
D3D12_MESSAGE_ID_CANNOT_USE_COMMAND_RECORDER_WITHOUT_CURRENT_TARGET,
D3D12_MESSAGE_ID_CANNOT_CHANGE_COMMAND_RECORDER_TARGET_WHILE_RECORDING,
D3D12_MESSAGE_ID_COMMAND_POOL_SYNC,
D3D12_MESSAGE_ID_EVICT_UNDERFLOW,
D3D12_MESSAGE_ID_CREATE_META_COMMAND,
D3D12_MESSAGE_ID_LIVE_META_COMMAND,
D3D12_MESSAGE_ID_DESTROY_META_COMMAND,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_DST_RESOURCE,
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_SRC_RESOURCE,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE,
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_BUFFER,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_RESOURCE_DESC,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_UNSUPPORTED,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_DIMENSION,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_FLAGS,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_OFFSET,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_DIMENSION,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_FLAGS,
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_OUTOFMEMORY_RETURN,
D3D12_MESSAGE_ID_CANNOT_CREATE_GRAPHICS_AND_VIDEO_COMMAND_RECORDER,
D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_POSSIBLY_MISMATCHING_PROPERTIES,
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE,
D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INCOMPATIBLE_WITH_STRUCTURED_BUFFERS,
D3D12_MESSAGE_ID_COMPUTE_ONLY_DEVICE_OPERATION_UNSUPPORTED,
D3D12_MESSAGE_ID_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INVALID,
D3D12_MESSAGE_ID_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_INVALID,
D3D12_MESSAGE_ID_COPY_RAYTRACING_ACCELERATION_STRUCTURE_INVALID,
D3D12_MESSAGE_ID_DISPATCH_RAYS_INVALID,
D3D12_MESSAGE_ID_GET_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO_INVALID,
D3D12_MESSAGE_ID_CREATE_LIFETIMETRACKER,
D3D12_MESSAGE_ID_LIVE_LIFETIMETRACKER,
D3D12_MESSAGE_ID_DESTROY_LIFETIMETRACKER,
D3D12_MESSAGE_ID_DESTROYOWNEDOBJECT_OBJECTNOTOWNED,
D3D12_MESSAGE_ID_CREATE_TRACKEDWORKLOAD,
D3D12_MESSAGE_ID_LIVE_TRACKEDWORKLOAD,
D3D12_MESSAGE_ID_DESTROY_TRACKEDWORKLOAD,
D3D12_MESSAGE_ID_RENDER_PASS_ERROR,
D3D12_MESSAGE_ID_META_COMMAND_ID_INVALID,
D3D12_MESSAGE_ID_META_COMMAND_UNSUPPORTED_PARAMS,
D3D12_MESSAGE_ID_META_COMMAND_FAILED_ENUMERATION,
D3D12_MESSAGE_ID_META_COMMAND_PARAMETER_SIZE_MISMATCH,
D3D12_MESSAGE_ID_UNINITIALIZED_META_COMMAND,
```

```
D3D12_MESSAGE_ID_META_COMMAND_INVALID_GPU_VIRTUAL_ADDRESS,
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDLIST,
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDLIST,
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDLIST,
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDQUEUE,
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDQUEUE,
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDQUEUE,
D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONESTIMATOR,
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONESTIMATOR,
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONESTIMATOR,
D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONVECTORHEAP,
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONVECTORHEAP,
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONVECTORHEAP,
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOADS,
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOAD_PAIRS,
D3D12_MESSAGE_ID_OUT_OF_ORDER_TRACKED_WORKLOAD_PAIR,
D3D12_MESSAGE_ID_CANNOT_ADD_TRACKED_WORKLOAD,
D3D12_MESSAGE_ID_INCOMPLETE_TRACKED_WORKLOAD_PAIR,
D3D12_MESSAGE_ID_CREATE_STATE_OBJECT_ERROR,
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_ERROR,
D3D12_MESSAGE_ID_GET_SHADER_STACK_SIZE_ERROR,
D3D12_MESSAGE_ID_GET_PIPELINE_STACK_SIZE_ERROR,
D3D12_MESSAGE_ID_SET_PIPELINE_STACK_SIZE_ERROR,
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_SIZE_INVALID,
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_INVALID,
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_DRIVER_REPORTED_ISSUE,
D3D12_MESSAGE_ID_RENDER_PASS_INVALID_RESOURCE_BARRIER,
D3D12_MESSAGE_ID_RENDER_PASS_DISALLOWED_API_CALLED,
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_NEST_RENDER_PASSES,
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_END_WITHOUT_BEGIN,
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_CLOSE_COMMAND_LIST,
D3D12_MESSAGE_ID_RENDER_PASS_GPU_WORK_WHILE_SUSPENDED,
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_SUSPEND_RESUME,
D3D12_MESSAGE_ID_RENDER_PASS_NO_PRIOR_SUSPEND_WITHIN_EXECUTECOMMANDLISTS,
D3D12_MESSAGE_ID_RENDER_PASS_NO_SUBSEQUENT_RESUME_WITHIN_EXECUTECOMMANDLISTS,
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_COMMAND_QUEUE_MISMATCH,
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_NOT_SUPPORTED,
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_NO_ACCESS,
D3D12_MESSAGE_ID_RENDER_PASS_UNSUPPORTED_RESOLVE,
D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INVALID_RESOURCE_PTR,
D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_SIGNAL,
D3D12_MESSAGE_ID_WINDOWS7_FENCE_OUTOFORDER_WAIT,
D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_ESTIMATOR_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_VIDEO_CREATE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_ESTIMATE_MOTION_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_RESOLVE_MOTION_VECTOR_HEAP_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_HEAP_TYPE,
D3D12_MESSAGE_ID_SET_BACKGROUND_PROCESSING_MODE_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE_FOR_FEATURE_LEVEL,
D3D12_MESSAGE_ID_CREATE_VIDEOEXTENSIONCOMMAND,
D3D12_MESSAGE_ID_LIVE_VIDEOEXTENSIONCOMMAND,
D3D12_MESSAGE_ID_DESTROY_VIDEOEXTENSIONCOMMAND,
D3D12_MESSAGE_ID_INVALID_VIDEO_EXTENSION_COMMAND_ID,
D3D12_MESSAGE_ID_VIDEO_EXTENSION_COMMAND_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_NOT_UNIQUE_IN_DXIL_LIBRARY,
D3D12_MESSAGE_ID_VARIABLE_SHADING_RATE_NOT_ALLOWED_WITH_TIR,

D3D12_MESSAGE_ID_GEOMETRY_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_SHADING_RATE,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_SHADING_RATE_NOT_PERMITTED_BY_CAP,
D3D12_MESSAGE_ID_RSSETSHADING_RATE_INVALID_COMBINER,
D3D12_MESSAGE_ID_RSSETSHADINGRATEIMAGE_REQUIRES_TIER_2,
D3D12_MESSAGE_ID_RSSETSHADINGRATE_REQUIRES_TIER_1,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_FORMAT,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_ARRAY_SIZE,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_MIP_LEVEL,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_COUNT,
D3D12_MESSAGE_ID_SHADING_RATE_IMAGE_INCORRECT_SAMPLE_QUALITY,
D3D12_MESSAGE_ID_NON_RETAIL_SHADER_MODEL_WONT_VALIDATE,
```

```

D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_AS_ROOT_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_MS_ROOT_SIGNATURE_MISMATCH,
D3D12_MESSAGE_ID_ADD_TO_STATE_OBJECT_ERROR,
D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION_INVALID_ARGUMENT,
D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_MS_PSO_DESC_MISMATCH,
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_MS_INCOMPLETE_TYPE,
D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_AS_NOT_MS_MISMATCH,
D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_MS_NOT_PS_MISMATCH,
D3D12_MESSAGE_ID_NONZERO_SAMPLER_FEEDBACK_MIP_REGION_WITH_INCOMPATIBLE_FORMAT,
D3D12_MESSAGE_ID_CREATEGRAPHICSPIPELINESTATE_INPUTLAYOUT_SHADER_MISMATCH,
D3D12_MESSAGE_ID_EMPTY_DISPATCH,
D3D12_MESSAGE_ID_RESOURCE_FORMAT_REQUIRES_SAMPLER_FEEDBACK_CAPABILITY,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_MIP_REGION,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_DIMENSION,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_COUNT,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_SAMPLE_QUALITY,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_INVALID_LAYOUT,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_MAP_REQUIRES_UNORDERED_ACCESS_FLAG,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_NULL_ARGUMENTS,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_UAV_REQUIRES_SAMPLER_FEEDBACK_CAPABILITY,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_REQUIRES_FEEDBACK_MAP_FORMAT,
D3D12_MESSAGE_ID_CREATEMESHSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTOFMEMORY,
D3D12_MESSAGE_ID_CREATEMESHSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_INVALID_FORMAT,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_INVALID_MIP_LEVEL_COUNT,
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_SAMPLER_FEEDBACK_TRANSCODE_ARRAY_SIZE_MISMATCH,
D3D12_MESSAGE_ID_SAMPLER_FEEDBACK_CREATE_UAV_MISMATCHING_TARGETED_RESOURCE,
D3D12_MESSAGE_ID_CREATEMESHSHADER_OUTPUTTEXCEEDSMAXSIZE,
D3D12_MESSAGE_ID_CREATEMESHSHADER_GROUPSHAREDEXCEEDSMAXSIZE,
D3D12_MESSAGE_ID_VERTEX_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE,
D3D12_MESSAGE_ID_MESH_SHADER_OUTPUTTING_BOTH_VIEWPORT_ARRAY_INDEX_AND_SHADING_RATE_NOT_SUPPORTED_ON_DEVICE,
D3D12_MESSAGE_ID_CREATEMESHSHADER_MISMATCHEDASMSPAYLOADSIZE,
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_UNBOUNDED_STATIC_DESCRIPTORS,
D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_INVALIDSHADERBYTECODE,
D3D12_MESSAGE_ID_CREATEAMPLIFICATIONSHADER_OUTOFMEMORY,
D3D12_MESSAGE_ID_D3D12_MESSAGES_END
} ;

```

Constants

D3D12_MESSAGE_ID_UNKNOWN	
--------------------------	--

D3D12_MESSAGE_ID_STRING_FROM_APPLICATION	
--	--

D3D12_MESSAGE_ID_CORRUPTED_THIS	
---------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER1	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER2	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER3	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER4	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER5	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER6	
---------------------------------------	--

D3D12_MESSAGE_ID_CORRUPTED_PARAMETER7
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER8
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER9
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER10
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER11
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER12
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER13
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER14
D3D12_MESSAGE_ID_CORRUPTED_PARAMETER15
D3D12_MESSAGE_ID_CORRUPTED_MULTITHREADING
D3D12_MESSAGE_ID_MESSAGE_REPORTING_OUTOFMEMORY
D3D12_MESSAGE_ID_GETPRIVATEDATAMOREDATA
D3D12_MESSAGE_ID_SETPRIVATEDATA_INVALIDFREEDATA
D3D12_MESSAGE_ID_SETPRIVATEDATA_CHANGINGPARAMS
D3D12_MESSAGE_ID_SETPRIVATEDATA_OUTOFMEMORY
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_UNREGISTEREDFORMAT
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDESC
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDFORMAT
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDVIDEOPLANESLICE
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDPLANESLICE
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDDIMENSIONS
D3D12_MESSAGE_ID_CREATESHADERRESOURCEVIEW_INVALIDRESOURCE

D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNRECOGNIZEDFORMAT
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_UNSUPPORTEDFORMAT
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDESC
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDFORMAT
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDVIDEOPLANESLICE
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDPLANESLICE
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDDIMENSIONS
D3D12_MESSAGE_ID_CREATERENDERTARGETVIEW_INVALIDRESOURCE
D3D12_MESSAGE_ID_CREATEDDEPTHSTENCILVIEW_UNRECOGNIZEDFORMAT
D3D12_MESSAGE_ID_CREATEDDEPTHSTENCILVIEW_INVALIDDESC
D3D12_MESSAGE_ID_CREATEDDEPTHSTENCILVIEW_INVALIDFORMAT
D3D12_MESSAGE_ID_CREATEDDEPTHSTENCILVIEW_INVALIDDIMENSIONS
D3D12_MESSAGE_ID_CREATEDDEPTHSTENCILVIEW_INVALIDRESOURCE
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_OUTOFCMEMORY
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TOOMANYELEMENTS
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDFORMAT
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INCOMPATIBLEFORMAT
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOT

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDINPUTSLOTCLASS
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_STEPRATESLOTCLASSMISMATCH
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDDSLOTCLASSCHANGE
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDSTEPRATECHANGE
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_INVALIDALIGNMENT
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_DUPLICATESEMANTIC
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_UNPARSEABLEINPUTSIGNATURE
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_NULLSEMANTIC
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_MISSINGELEMENT
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_OUTOFCMEMORY
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERBYTECODE
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDSHADERTYPE
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_OUTOFCMEMORY
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERBYTECODE
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDSHADERTYPE
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTOFCMEMORY
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERBYTECODE
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSHADERTYPE

D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDNUMENTRIES	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSTREAMSTRIDEUNUSED	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_OUTPUTSLOT0EXPECTED	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSLOT	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_ONLYONEELEMENTPERSLOT	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDCOMPONENTCOUNT	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDSTARTCOMPONENTANDCOMPONENTCOUNT	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDGAPDEFINITION	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_REPEATODOPUT	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_INVALIDOUTPUTSTREAMSTRIDE	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGSEMANTIC	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MASKMISMATCH	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_CANTHAVEONLYGAPS	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DECLTOOCOMPLEX	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_MISSINGOUTPUTSIGNATURE	
D3D12_MESSAGE_ID_CREATEPIXELSHADER_OUTOFMEMORY	
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERBYTECODE	
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDSHADERTYPE	

D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDFILL MODE	
D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDCULL MODE	
D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDDEPT HBIASCLAMP	
D3D12_MESSAGE_ID_CREATEASTERIZERSTATE_INVALIDDSOP ESCALEDEPTHBIAS	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDD EPTHWRITEMASK	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDD EPTHFUNC	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFR ONTFACE_STENCILFAILOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFR ONTFACE_STENCILZFAILOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFR ONTFACE_STENCILPASSOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDFR ONTFACE_STENCILFUNC	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDIB ACKFACE_STENCILFAILOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDIB ACKFACE_STENCILZFAILOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDIB ACKFACE_STENCILPASSOP	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_INVALIDIB ACKFACE_STENCILFUNC	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEN D	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDSRCBLEND ALPHA	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDDESTBLEN DALPHA	

D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDBLENDOP_ALPHA	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDRENDERTARGETWRITEMASK	
D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_INVALID	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_NOT_SET	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_ROOT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_NOT_SET	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_STRIDE_TOO_SMALL	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_BUFFER_TOO_SMALL	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_NOT_SET	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_FORMAT_INVALID	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_BUFFER_TOO_SMALL	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INVALID_PRIMITIVE_TOPOLOGY	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_VERTEX_STRIDE_UNALIGNED	
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_INDEX_OFFSET_UNALIGNED	
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_AT_FAULT	
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_POSSIBLY_AT_FAULT	
D3D12_MESSAGE_ID_DEVICE_REMOVAL_PROCESS_NOT_AT_FAULT	
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TRAILING_DIGIT_IN_SEMANTIC	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_TRAILING_DIGIT_IN_SEMANTIC	

D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_TYPE_MISMATCH	
D3D12_MESSAGE_ID_CREATEINPUTLAYOUT_EMPTY_LAYOUT	
D3D12_MESSAGE_ID_LIVE_OBJECT_SUMMARY	
D3D12_MESSAGE_ID_LIVE_DEVICE	
D3D12_MESSAGE_ID_LIVE_SWAPCHAIN	
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILVIEW_INVALIDFLAGS	
D3D12_MESSAGE_ID_CREATEVERTEXSHADER_INVALIDCLASSLINKAGE	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADER_INVALIDCLASSLINKAGE	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_INVALIDSTREAMTORASTERIZER	
D3D12_MESSAGE_ID_CREATEPIXELSHADER_INVALIDCLASSLINKAGE	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_INVALIDSTREAM	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_UNEXPECTEDENTRIES	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_UNEXPECTEDSTRIDES	
D3D12_MESSAGE_ID_CREATEGEOMETRYSHADERWITHSTREAM_OUTPUT_INVALIDNUMSTRIDES	
D3D12_MESSAGE_ID_CREATEHULLSHADER_OUTOFMEMORY	
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADER_BYTECODE	
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDSHADER_TYPE	
D3D12_MESSAGE_ID_CREATEHULLSHADER_INVALIDCLASSLINKAGE	
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_OUTOFMEMORY	
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADER_BYTECODE	

D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDSHADERTYPE
D3D12_MESSAGE_ID_CREATEDOMAINSHADER_INVALIDCLASLINKAGE
D3D12_MESSAGE_ID_RESOURCE_UNMAP_NOTMAPPED
D3D12_MESSAGE_ID_DEVICE_CHECKFEATURESUPPORT_MISMATCHED_DATA_SIZE
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_OUTOFGMEMORY
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDSHADERBYTECODE
D3D12_MESSAGE_ID_CREATECOMPUTESHADER_INVALIDCLASSLINKAGE
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEFLOATOPSNOTSUPPORTED
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDRESOURCE
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDESC
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFORMAT

D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDVIDEOPLANESLICE	
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDPLANESLICE	
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDDIMENSIONS	
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_UNRECOGNIZEDFORMAT	
D3D12_MESSAGE_ID_CREATEUNORDEREDACCESSVIEW_INVALIDFLAGS	
D3D12_MESSAGE_ID_CREATERASTERIZERSTATE_INVALIDFORCEDSAMPLECOUNT	
D3D12_MESSAGE_ID_CREATEBLENDSTATE_INVALIDLOGICOPS	
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWITHSTREAMOUTPUT_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_DOUBLEEXTENSIONSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEVERTEXSHADER_UAVSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEHULLSHADER_UAVSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEDOMAINSHADER_UAVSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADER_UAVSNOTSUPPORTED	

D3D12_MESSAGE_ID_DEVICE_CREATEGEOMETRYSHADERWIT HSTREAMOUTPUT_UAVSNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATEPIXELSHADER_UAVSNO TSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CREATECOMPUTESHADER_UAV SNOTSUPPORTED	
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_INVALIDSOURCER ECT	
D3D12_MESSAGE_ID_DEVICE_CLEARVIEW_EMPTYRECT	
D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_INVALID_PARA METER	
D3D12_MESSAGE_ID_COPYTILEMAPPINGS_INVALID_PARAME TER	
D3D12_MESSAGE_ID_CREATEDevice_INVALIDARGS	
D3D12_MESSAGE_ID_CREATEDevice_WARNING	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_TYPE	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_NULL_POINTER	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SUBRESO URCE	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_RESERVED_BITS	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISSING_BIND_FL AGS	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_M ISC_FLAGS	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MATCHING_STATE S	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBIN ATION	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_BEFORE_AFTER_MI SMATCH	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_RESOUR CE	
D3D12_MESSAGE_ID_RESOURCE_BARRIER_SAMPLE_COUNT	

D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMBINED_FLAGS
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAGS_FORMAT
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_SPLIT_BARRIER
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_END
D3D12_MESSAGE_ID_RESOURCE_BARRIER_UNMATCHED_BEGIN
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_FLAG
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_COMMAND_LIST_TYPE
D3D12_MESSAGE_ID_INVALID_SUBRESOURCE_STATE
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CONTENTION
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_RESET_BUNDLE
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_CANNOT_RESET
D3D12_MESSAGE_ID_COMMAND_LIST_OPEN
D3D12_MESSAGE_ID_INVALID_BUNDLE_API
D3D12_MESSAGE_ID_COMMAND_LIST_CLOSED
D3D12_MESSAGE_ID_WRONG_COMMAND_ALLOCATOR_TYPE
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_SYNC
D3D12_MESSAGE_ID_COMMAND_LIST_SYNC
D3D12_MESSAGE_ID_SET_DESCRIPTOR_HEAP_INVALID
D3D12_MESSAGE_ID_CREATE_COMMANDQUEUE
D3D12_MESSAGE_ID_CREATE_COMMANDALLOCATOR

D3D12_MESSAGE_ID_CREATE_PIPELINESTATE
D3D12_MESSAGE_ID_CREATE_COMMANDLIST12
D3D12_MESSAGE_ID_CREATE_RESOURCE
D3D12_MESSAGE_ID_CREATE_DESCRIPTORHEAP
D3D12_MESSAGE_ID_CREATE_ROOTSIGNATURE
D3D12_MESSAGE_ID_CREATE_LIBRARY
D3D12_MESSAGE_ID_CREATE_HEAP
D3D12_MESSAGE_ID_CREATE_MONITOREDFENCE
D3D12_MESSAGE_ID_CREATE_QUERYHEAP
D3D12_MESSAGE_ID_CREATE_COMMANDSIGNATURE
D3D12_MESSAGE_ID_LIVE_COMMANDQUEUE
D3D12_MESSAGE_ID_LIVE_COMMANDALLOCATOR
D3D12_MESSAGE_ID_LIVE_PIPELINESTATE
D3D12_MESSAGE_ID_LIVE_COMMANDLIST12
D3D12_MESSAGE_ID_LIVE_RESOURCE
D3D12_MESSAGE_ID_LIVE_DESCRIPTORHEAP
D3D12_MESSAGE_ID_LIVE_ROOTSIGNATURE
D3D12_MESSAGE_ID_LIVE_LIBRARY
D3D12_MESSAGE_ID_LIVE_HEAP
D3D12_MESSAGE_ID_LIVE_MONITOREDFENCE
D3D12_MESSAGE_ID_LIVE_QUERYHEAP
D3D12_MESSAGE_ID_LIVE_COMMANDSIGNATURE
D3D12_MESSAGE_ID_DESTROY_COMMANDQUEUE
D3D12_MESSAGE_ID_DESTROY_COMMANDALLOCATOR
D3D12_MESSAGE_ID_DESTROY_PIPELINESTATE
D3D12_MESSAGE_ID_DESTROY_COMMANDLIST12

D3D12_MESSAGE_ID_DESTROY_RESOURCE
D3D12_MESSAGE_ID_DESTROY_DESCRIPTORHEAP
D3D12_MESSAGE_ID_DESTROY_ROOTSIGNATURE
D3D12_MESSAGE_ID_DESTROY_LIBRARY
D3D12_MESSAGE_ID_DESTROY_HEAP
D3D12_MESSAGE_ID_DESTROY_MONITOREDFENCE
D3D12_MESSAGE_ID_DESTROY_QUERYHEAP
D3D12_MESSAGE_ID_DESTROY_COMMANDSIGNATURE
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDIMENSION_S
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDMISCFLAGS
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDARG_RETURN
D3D12_MESSAGE_ID_CREATERESOURCE_OUTOFMEMORY_RETURN
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDESC
D3D12_MESSAGE_ID_POSSIBLY_INVALID_SUBRESOURCE_STATE
D3D12_MESSAGE_ID_INVALID_USE_OF_NON_RESIDENT_RESOURCE
D3D12_MESSAGE_ID_POSSIBLE_INVALID_USE_OF_NON_RESIDENT_RESOURCE
D3D12_MESSAGE_ID_BUNDLE_PIPELINE_STATE_MISMATCH
D3D12_MESSAGE_ID_PRIMITIVE_TOPOLOGY_MISMATCH_PIPELINE_STATE
D3D12_MESSAGE_ID_RENDER_TARGET_FORMAT_MISMATCH_PIPELINE_STATE
D3D12_MESSAGE_ID_RENDER_TARGET_SAMPLE_DESC_MISMATCH_PIPELINE_STATE
D3D12_MESSAGE_ID_DEPTH_STENCIL_FORMAT_MISMATCH_PIPELINE_STATE

D3D12_MESSAGE_ID_DEPTH_STENCIL_SAMPLE_DESC_MISMATCH_PIPELINE_STATE
D3D12_MESSAGE_ID_CREATESHADER_INVALIDBYTECODE
D3D12_MESSAGE_ID_CREATEHEAP_NULLDESC
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDSIZE
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDHEAPTYPE
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMEMORYPOOL
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDPROPERTIES
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDALIGNMENT
D3D12_MESSAGE_ID_CREATEHEAP_UNRECOGNIZEDMISCFLAGS
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDMISCFLAGS
D3D12_MESSAGE_ID_CREATEHEAP_INVALIDARG_RETURN
D3D12_MESSAGE_ID_CREATEHEAP_OUTOFMEMORY_RETURN
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLHEAPPROPERTIES
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPTYPE
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDCPUPAGEPROPERTIES
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDMEMORYPOOL
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPPROPERTIES
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_UNRECOGNIZEDHEAPMISCFLAGS
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_INVALIDHEAPMISCFLAGS

D3D12_MESSAGE_ID_CREATRESOURCEANDHEAP_INVALIDARG_RETURN
D3D12_MESSAGE_ID_CREATRESOURCEANDHEAP_OUTOFMEMORY_RETURN
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_UNREGISTEREDHEAPTYPE
D3D12_MESSAGE_ID_GETCUSTOMHEAPPROPERTIES_INVALIDHEAPTYPE
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_INVALID_DESC
D3D12_MESSAGE_ID_INVALID_DESCRIPTOR_HANDLE
D3D12_MESSAGE_ID_CREATRASTERIZERSTATE_INVALID_CO_NSERVATIVEASTERMODE
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_RESOURCE
D3D12_MESSAGE_ID_CREATE_CONSTANT_BUFFER_VIEW_INVALID_DESC
D3D12_MESSAGE_ID_CREATE_UNORDEREDACCESS_VIEW_INVALID_COUNTER_USAGE
D3D12_MESSAGE_ID_COPY_DESCRIPTOR_INVALID_RANGES
D3D12_MESSAGE_ID_COPY_DESCRIPTOR_WRITE_ONLY_DESCRIPTOR
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_FORMAT_NOT_UNKNOWN
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INVALID_RENDER_TARGET_COUNT
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_VERTEX_SHADER_NOT_SET
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_INPUT_LAYOUT_NOT_SET
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_HS_DS_SIGNATURE_MISMATCH
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_REGISTERINDEX
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_SHADER_LINKAGE_COMPONENTTYPE

D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Linkage_RegisterMask
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Linkage_SystemValue
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Linkage_NeverWritten_AlwaysReads
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Linkage_MinPrecision
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Linkage_SemanticName_Not_Found
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_HS_Xor_DS_Mismatch
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Hull_Shader_Input_Topo_Mismatch
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_HS_DS_Control_Point_Count_Mismatch
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_HS_DS_Tessellator_Domain_Mismatch
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Use_of_Center_MultiSample_Pattern
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_Use_of_Forced_Sample_Count
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_PrimitiveTopology
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Invalid_SystemValue
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Om_Dual_Source_Blending_Can_Only_Have_Render_Target_0
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Om_Renderer_Target_Does_Not_Support_Blending
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Ps_Output_Type_Mismatch
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Om_Renderer_Target_Does_Not_Support_Logic_Ops
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_RenderTargetView_Not_Set

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_DEPT H_STENCILVIEW_NOT_SET	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_GS_IN PUT_PRIMITIVE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_POSIT ION_NOT_PRESENT	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_MISSI NG_ROOT_SIGNATURE_FLAGS	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVAL ID_INDEX_BUFFER_PROPERTIES	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_INVAL ID_SAMPLE_DESC	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_HS_R OOT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_DS_R OOT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_VS_R OOT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_GS_R OOT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_PS_RO OT_SIGNATURE_MISMATCH	
D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_MISSI NG_ROOT_SIGNATURE	
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_OPEN_BUNDLE	
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_DESCRIPTOR_HEAP_ MISMATCH	
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_TYPE	
D3D12_MESSAGE_ID_DRAW_EMPTY_SCISSOR_RECTANGLE	
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_BLOB_NOT_ FOUND	
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_DESERIALIZ E_FAILED	
D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_INVALID_C ONFIGURATION	

D3D12_MESSAGE_ID_CREATE_ROOT_SIGNATURE_NOT_SUPPORTED_ON_DEVICE
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLRESOURCEPROPERTIES
D3D12_MESSAGE_ID_CREATERESOURCEANDHEAP_NULLHEAP
D3D12_MESSAGE_ID_GETRESOURCEALLOCATIONINFO_INVALIDRDESCS
D3D12_MESSAGE_ID_MAKERESIDENT_NULLOBJECTARRAY
D3D12_MESSAGE_ID_EVICT_NULLOBJECTARRAY
D3D12_MESSAGE_ID_SET_DESCRIPTOR_TABLE_INVALID
D3D12_MESSAGE_ID_SET_ROOT_CONSTANT_INVALID
D3D12_MESSAGE_ID_SET_ROOT_CONSTANT_BUFFER_VIEW_INVALID
D3D12_MESSAGE_ID_SET_ROOT_SHADER_RESOURCE_VIEW_INVALID
D3D12_MESSAGE_ID_SET_ROOT_UNORDERED_ACCESS_VIEW_INVALID
D3D12_MESSAGE_ID_SET_VERTEX_BUFFERS_INVALID_DESC
D3D12_MESSAGE_ID_SET_INDEX_BUFFER_INVALID_DESC
D3D12_MESSAGE_ID_SET_STREAM_OUTPUT_BUFFERS_INVALID_DESC
D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDDIMENSIONALITY
D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZED_LAYOUT
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDDIMENSIONALITY
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDALIGNMENT
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDMIPLEVELS
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDSAMPLEDESC

D3D12_MESSAGE_ID_CREATEROLESOURCE_INVALIDROLE
D3D12_MESSAGE_ID_SET_INDEX_BUFFER_INVALID
D3D12_MESSAGE_ID_SET_VERTEX_BUFFERS_INVALID
D3D12_MESSAGE_ID_SET_STREAM_OUTPUT_BUFFERS_INVALID
D3D12_MESSAGE_ID_SET_RENDER_TARGETS_INVALID
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_INVALID_PARAMETERS
D3D12_MESSAGE_ID_BEGIN_END_QUERY_INVALID_PARAMETERS
D3D12_MESSAGE_ID_CLOSE_COMMAND_LIST_OPEN_QUERY
D3D12_MESSAGE_ID_RESOLVE_QUERY_DATA_INVALID_PARAMETERS
D3D12_MESSAGE_ID_SET_PREDICATION_INVALID_PARAMETERS
D3D12_MESSAGE_ID_TIMESTAMPS_NOT_SUPPORTED
D3D12_MESSAGE_ID_CREATEROLESOURCE_UNRECOGNIZEDFORMAT
D3D12_MESSAGE_ID_CREATEROLESOURCE_INVALIDFORMAT
D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDSUBRESOURCE_RANGE
D3D12_MESSAGE_ID_GETCOPYABLELAYOUT_INVALIDBASEOFFSET
D3D12_MESSAGE_ID_RESOURCE_BARRIER_INVALID_HEAP
D3D12_MESSAGE_ID_CREATE_SAMPLER_INVALID
D3D12_MESSAGE_ID_CREATECOMMANDSIGNATURE_INVALID
D3D12_MESSAGE_ID_EXECUTE_INDIRECT_INVALID_PARAMETERS
D3D12_MESSAGE_ID_GETGPUVIRTUALADDRESS_INVALID_RESOURCE_DIMENSION
D3D12_MESSAGE_ID_CREATEROLESOURCE_INVALIDCLEARVALUE

D3D12_MESSAGE_ID_CREATERESOURCE_UNRECOGNIZEDCLEARVALUEFORMAT
D3D12_MESSAGE_ID_CREATERESOURCE_INVALIDCLEARVALUEFORMAT
D3D12_MESSAGE_ID_CREATERESOURCE_CLEARVALUEDENORMFLUSH
D3D12_MESSAGE_ID_CLEARRENDERTARGETVIEW_MISMATCHINGCLEARVALUE
D3D12_MESSAGE_ID_CLEARDEPTHSTENCILVIEW_MISMATCHINGCLEARVALUE
D3D12_MESSAGE_ID_MAP_INVALIDHEAP
D3D12_MESSAGE_ID_UNMAP_INVALIDHEAP
D3D12_MESSAGE_ID_MAP_INVALIDRESOURCE
D3D12_MESSAGE_ID_UNMAP_INVALIDRESOURCE
D3D12_MESSAGE_ID_MAP_INVALIDSUBRESOURCE
D3D12_MESSAGE_ID_UNMAP_INVALIDSUBRESOURCE
D3D12_MESSAGE_ID_MAP_INVALIDRANGE
D3D12_MESSAGE_ID_UNMAP_INVALIDRANGE
D3D12_MESSAGE_ID_MAP_INVALIDDATAPORTER
D3D12_MESSAGE_ID_MAP_INVALIDARG_RETURN
D3D12_MESSAGE_ID_MAP_OUTOFCMEMORY_RETURN
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_BUNDLENOTSUPPORTED
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_COMMANDLISTMISMATCH
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_OPENCOMMANDLIST
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_FAILEDCOMMANDLIST
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLDST
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDDSTRESOURCEDIMENSION

D3D12_MESSAGE_ID_COPYBUFFERREGION_DSTRANGEOUTO FBOUNDS	
D3D12_MESSAGE_ID_COPYBUFFERREGION_NULLSRC	
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDSRCRES OURCEDIMENSION	
D3D12_MESSAGE_ID_COPYBUFFERREGION_SRCRANGEOUTO FBOUNDS	
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALIDCOPYFL AGS	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLDST	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZE DDSTTYPE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRES OURCEDIMENSION	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRES OURCE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTSUB RESOURCE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTOFF SET	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZE DDSTFORMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTFO RMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDIM ENSIONS	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTRO WPITCH	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTPLA CEMENT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTDSP LACEDFOOTPRINTFORMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_DSTREGIONOU TOFBOUNDS	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_NULLSRC	

D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZE DSRCTYPE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRES OURCEDIMENSION	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRES OURCE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCSUB RESOURCE	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCOFF SET	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_UNRECOGNIZE DSRCFORMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCFO RMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDI MENSIONS	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCRO WPITCH	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCPLA CEMENT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCDSP LACEDFOOTPRINTFORMAT	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_SRCREGIONOU TOFBOUNDS	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDDSTCO ORDINATES	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDSRCBO X	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_FORMATMISMA TCH	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_EMPTYBOX	
D3D12_MESSAGE_ID_COPYTEXTUREREGION_INVALIDCOPYF LAGS	
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SUBL E_SOURCE_INDEX	

D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_FORMAT
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_MISMATCH
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALID_SAMPLE_COUNT
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_INVALID_SHADER
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_CS_ROOT_SIGNATURE_MISMATCH
D3D12_MESSAGE_ID_CREATECOMPUTPIPELINESTATE_MISSING_ROOT_SIGNATURE
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALIDCACHE_DBLOB
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBA_DAPTERMISMATCH
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDRIVERVERSIONMISMATCH
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBDE_SCMMISMATCH
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CACHEDBLOBIGNORED
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDHEAP
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDRESOURCE
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDBOX
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_INVALIDSUBRESOURCE
D3D12_MESSAGE_ID_WRITETOSUBRESOURCE_EMPTYBOX
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDHEAP
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDRESOURCE
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDBOX

D3D12_MESSAGE_ID_READFROMSUBRESOURCE_INVALIDSUBRESOURCE	
D3D12_MESSAGE_ID_READFROMSUBRESOURCE_EMPTYBOX	
D3D12_MESSAGE_ID_TOO_MANY_NODES_SPECIFIED	
D3D12_MESSAGE_ID_INVALID_NODE_INDEX	
D3D12_MESSAGE_ID_GETTHEAPPROPERTIES_INVALIDRESOURCE	
D3D12_MESSAGE_ID_NODE_MASK_MISMATCH	
D3D12_MESSAGE_ID_COMMAND_LIST_OUTOFCMEMORY	
D3D12_MESSAGE_ID_COMMAND_LIST_MULTIPLE_SWAPCHAIN_BUFFER_REFERENCES	
D3D12_MESSAGE_ID_COMMAND_LIST_TOO_MANY_SWAPCHAIN_REFERENCES	
D3D12_MESSAGE_ID_COMMAND_QUEUE_TOO_MANY_SWAPCHAIN_REFERENCES	
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_WRONGSWAPCHAINBUFFERREFERENCE	
D3D12_MESSAGE_ID_COMMAND_LIST_SETRENDERTARGETS_INVALIDNUMRENDERTARGETS	
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_TYPE	
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_FLAGS	
D3D12_MESSAGE_ID_CREATESHAREDRESOURCE_INVALIDFLAGS	
D3D12_MESSAGE_ID_CREATESHAREDRESOURCE_INVALIDFORMAT	
D3D12_MESSAGE_ID_CREATESHAREDHEAP_INVALIDFLAGS	
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_UNRECOGNIZEDPROPERTIES	
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDSIZE	
D3D12_MESSAGE_ID_REFLECTSHAREDPROPERTIES_INVALIDOBJECT	
D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDOBJECT	

D3D12_MESSAGE_ID_KEYEDMUTEX_INVALIDKEY
D3D12_MESSAGE_ID_KEYEDMUTEX_WRONGSTATE
D3D12_MESSAGE_ID_CREATE_QUEUE_INVALID_PRIORITY
D3D12_MESSAGE_ID_OBJECT_DELETED_WHILE_STILL_IN_USE
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_FLAGS
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_HAS_NO_RESOURCE
D3D12_MESSAGE_ID_COMMAND_LIST_DRAW_RENDER_TARGET_DELETED
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelinestate_all_render_targets_have_unknown_format
D3D12_MESSAGE_ID_HEAP_ADDRESS_RANGE_INTERSECTS_MULTIPLE_BUFFERS
D3D12_MESSAGE_ID_EXECUTECOMMANDLISTS_GPU_WRITEBACK_RESOURCE_MAPPED
D3D12_MESSAGE_ID_UNMAP_RANGE_NOT_EMPTY
D3D12_MESSAGE_ID_MAP_INVALID_NULLRANGE
D3D12_MESSAGE_ID_UNMAP_INVALID_NULLRANGE
D3D12_MESSAGE_ID_NO_GRAPHICS_API_SUPPORT
D3D12_MESSAGE_ID_NO_COMPUTE_API_SUPPORT
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_RESOURCE_FLAGS_NOT_SUPPORTED
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_ARGUMENT_UNINITIALIZED
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_HEAP_INDEX_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TABLE_REGISTER_INDEX_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_UNINITIALIZED
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_DESCRIPTOR_TYPE_MISMATCH

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SRV_RESOURCE_DIMENSION_MISMATCH
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UAV_RESOURCE_DIMENSION_MISMATCH
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INCOMPATIBLE_RESOURCE_STATE
D3D12_MESSAGE_ID_COPYRESOURCE_NULLDST
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDDSTRESOURCE
D3D12_MESSAGE_ID_COPYRESOURCE_NULLSRC
D3D12_MESSAGE_ID_COPYRESOURCE_INVALIDSRCRESOURCE
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLDST
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDDSTRESOURCEx
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_NULLSRC
D3D12_MESSAGE_ID_RESOLVESUBRESOURCE_INVALIDSRCRESOURCEx
D3D12_MESSAGE_ID_PIPELINE_STATE_TYPE_MISMATCH
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_NOT_SET
D3D12_MESSAGE_ID_COMMAND_LIST_DISPATCH_ROOT_SIGNATURE_MISMATCH
D3D12_MESSAGE_ID_RESOURCE_BARRIER_ZERO_BARRIERS
D3D12_MESSAGE_ID_BEGIN_END_EVENT_MISMATCH
D3D12_MESSAGE_ID_RESOURCE_BARRIER_POSSIBLE_BEFORE_AFTER_MISMATCH
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_BEGIN_END
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_INVALID_RESOURCE
D3D12_MESSAGE_ID_USE_OF_ZERO_REFCOUNT_OBJECT
D3D12_MESSAGE_ID_OBJECT_EVICTED_WHILE_STILL_IN_USE

D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_ROOT_DESC RIPTOR_ACCESS_OUT_OF_BOUNDS	
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_INVALIDLIBRARYBLOB	
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_DRIVERSVERSIONMISMATCH	
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_ADAPTERVERSIONMISMATCH	
D3D12_MESSAGE_ID_CREATEPIPELINELIBRARY_UNSUPPORTED	
D3D12_MESSAGE_ID_CREATE_PIPELINELIBRARY	
D3D12_MESSAGE_ID_LIVE_PIPELINELIBRARY	
D3D12_MESSAGE_ID_DESTROY_PIPELINELIBRARY	
D3D12_MESSAGE_ID_STOREPIPELINE_NONAME	
D3D12_MESSAGE_ID_STOREPIPELINE_DUPLICATENAME	
D3D12_MESSAGE_ID_LOADPIPELINE_NAMENOTFOUND	
D3D12_MESSAGE_ID_LOADPIPELINE_INVALIDDESC	
D3D12_MESSAGE_ID_PIPELINELIBRARY_SERIALIZE_NOTENOUGHMEMORY	
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineSTATE_PS_OUTPUT_RT_OUTPUT_MISMATCH	
D3D12_MESSAGE_ID_SETEVENTONMULTIPLEFENCECOMPLETION_INVALIDFLAGS	
D3D12_MESSAGE_ID_CREATE_QUEUE_VIDEO_NOT_SUPPORTED	
D3D12_MESSAGE_ID_CREATE_COMMAND_ALLOCATOR_VIDEO_NOT_SUPPORTED	
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_VIDEO_DECODE_STATISTICS_NOT_SUPPORTED	
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDLIST	
D3D12_MESSAGE_ID_CREATE_VIDEODECODER	
D3D12_MESSAGE_ID_CREATE_VIDEODECODESTREAM	

D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDLIST
D3D12_MESSAGE_ID_LIVE_VIDEODECODER
D3D12_MESSAGE_ID_LIVE_VIDEODECODESTREAM
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDLIST
D3D12_MESSAGE_ID_DESTROY_VIDEODECODER
D3D12_MESSAGE_ID_DESTROY_VIDEODECODESTREAM
D3D12_MESSAGE_ID_DECODE_FRAME_INVALID_PARAMETERS
D3D12_MESSAGE_ID_DEPRECATED_API
D3D12_MESSAGE_ID_RESOURCE_BARRIER_MISMATCHING_COMMAND_LIST_TYPE
D3D12_MESSAGE_ID_COMMAND_LIST_DESCRIPTOR_TABLE_NOT_SET
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_CONSTANT_BUFFER_VIEW_NOT_SET
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_SHADER_RESOURCE_VIEW_NOT_SET
D3D12_MESSAGE_ID_COMMAND_LIST_ROOT_UNORDERED_ACCESS_VIEW_NOT_SET
D3D12_MESSAGE_ID_DISCARD_INVALID_SUBRESOURCE_RANGE
D3D12_MESSAGE_ID_DISCARD_ONE_SUBRESOURCE_FOR_MIPMAPS_WITH_RECTS
D3D12_MESSAGE_ID_DISCARD_NO_RECTS_FOR_NON_TEXTURE2D
D3D12_MESSAGE_ID_COPY_ON_SAME_SUBRESOURCE
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PAGEABLE
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_UNSUPPORTED
D3D12_MESSAGE_ID_STATIC_DESCRIPTOR_INVALID_DESCRIPTOR_CHANGE

D3D12_MESSAGE_ID_DATA_STATIC_DESCRIPTOR_INVALID_DATA_CHANGE
D3D12_MESSAGE_ID_DATA_STATIC_WHILE_SET_AT_EXECUTE_DESCRIPTOR_INVALID_DATA_CHANGE
D3D12_MESSAGE_ID_EXECUTE_BUNDLE_STATIC_DESCRIPTOR_DATA_STATIC_NOT_SET
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_ACCESS_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_SAMPLER_MODE_MISMATCH
D3D12_MESSAGE_ID_CREATE_FENCE_INVALID_FLAGS
D3D12_MESSAGE_ID_RESOURCE_BARRIER_DUPLICATE_SUBRESOURCE_TRANSITIONS
D3D12_MESSAGE_ID_SETRESIDENCYPRIORITY_INVALID_PRIORITY
D3D12_MESSAGE_ID_CREATE_DESCRIPTOR_HEAP_LARGE_NUM_DESCRIPORS
D3D12_MESSAGE_ID_BEGIN_EVENT
D3D12_MESSAGE_ID_END_EVENT
D3D12_MESSAGE_ID_CREATEDevice_DEBUG_LAYER_STARTUP_OPTIONS
D3D12_MESSAGE_ID_CREATEDEPTHSTENCILSTATE_DEPTHBONDSTEST_UNSUPPORTED
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_DUPLICATE_SUBOBJECT
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_UNKNOWN_SUBOBJECT
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_ZERO_SIZE_STREAM
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_INVALID_STREAM
D3D12_MESSAGE_ID_CREATEPIPELINESTATE_CANNOT_DEDUCE_TYPE
D3D12_MESSAGE_ID_COMMAND_LIST_STATIC_DESCRIPTOR_RESOURCE_DIMENSION_MISMATCH

D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_PRIVILEGE_FOR_GLOBAL_REALTIME
D3D12_MESSAGE_ID_CREATE_COMMAND_QUEUE_INSUFFICIENT_HARDWARE_SUPPORT_FOR_GLOBAL_REALTIME
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_ARCHITECTURE
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DST
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE_DIMENSION
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DST_RANGE_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_SRC
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE_DIMENSION
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_SRC_RANGE_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_OFFSET_ALIGNMENT
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_RESOURCES
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_NULL_DEPENDENT_SUBRESOURCE_RANGES
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_RESOURCE
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DEPENDENT_SUBRESOURCE_RANGE
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_SUBRESOURCE_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_DEPENDENT_RANGE_OUT_OF_BOUNDS
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_ZERO_DEPENDENCIES
D3D12_MESSAGE_ID_DEVICE_CREATE_SHARED_HANDLE_INVALIDARG
D3D12_MESSAGE_ID_DESCRIPTOR_HANDLE_WITH_INVALID_RESOURCE

D3D12_MESSAGE_ID_SETDEPTHBOUNDS_INVALIDARGS
D3D12_MESSAGE_ID_GPU_BASED_VALIDATION_RESOURCE_STATE_IMPRECISE
D3D12_MESSAGE_ID_COMMAND_LIST_PIPELINE_STATE_NOT_SET
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_Shader_Model_MISMATCH
D3D12_MESSAGE_ID_OBJECT_ACCESSED_WHILE_STILL_IN_USE
D3D12_MESSAGE_ID_PROGRAMMABLE_MSAA_UNSUPPORTED
D3D12_MESSAGE_ID_SETSAMPLEPOSITIONS_INVALIDARGS
D3D12_MESSAGE_ID_RESOLVESUBRESOURCEREGION_INVALID_RECT
D3D12_MESSAGE_ID_CREATE_VIDEODECODECOMMANDQUEUE
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDLIST
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSCOMMANDQUEUE
D3D12_MESSAGE_ID_LIVE_VIDEODECODECOMMANDQUEUE
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDLIST
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSCOMMANDQUEUE
D3D12_MESSAGE_ID_DESTROY_VIDEODECODECOMMANDQUEUE
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSCOMMANDQUEUE
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSOR
D3D12_MESSAGE_ID_CREATE_VIDEOPROCESSSTREAM
D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSOR

D3D12_MESSAGE_ID_LIVE_VIDEOPROCESSSTREAM
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSOR
D3D12_MESSAGE_ID_DESTROY_VIDEOPROCESSSTREAM
D3D12_MESSAGE_ID_PROCESS_FRAME_INVALID_PARAMETERS
D3D12_MESSAGE_ID_COPY_INVALIDLAYOUT
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION
D3D12_MESSAGE_ID_CREATE_CRYPTO_SESSION_POLICY
D3D12_MESSAGE_ID_CREATE_PROTECTED_RESOURCE_SESSION
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION
D3D12_MESSAGE_ID_LIVE_CRYPTO_SESSION_POLICY
D3D12_MESSAGE_ID_LIVE_PROTECTED_RESOURCE_SESSION
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION
D3D12_MESSAGE_ID_DESTROY_CRYPTO_SESSION_POLICY
D3D12_MESSAGE_ID_DESTROY_PROTECTED_RESOURCE_SESSION
D3D12_MESSAGE_ID_PROTECTED_RESOURCE_SESSION_UNSUPPORTED
D3D12_MESSAGE_ID_FENCE_INVALIDOPERATION
D3D12_MESSAGE_ID_CREATEQUERY_HEAP_COPY_QUEUE_TIMESTAMPS_NOT_SUPPORTED
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_DEFERRED
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMFIRSTUSE
D3D12_MESSAGE_ID_SAMPLEPOSITIONS_MISMATCH_RECORDTIME_ASSUMEDFROMCLEAR
D3D12_MESSAGE_ID_CREATE_VIDEODECODERHEAP
D3D12_MESSAGE_ID_LIVE_VIDEODECODERHEAP

D3D12_MESSAGE_ID_DESTROY_VIDEODECODERHEAP
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDARG_RETURNS
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_OUTOFMEMORY_RETURN
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDADDRESS
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_INVALIDHANDLE
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_DEST
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_MODE
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_INVALID_ALI_GNMENT
D3D12_MESSAGE_ID_WRITEBUFFERIMMEDIATE_NOT_SUPPORTED
D3D12_MESSAGE_ID_SETVIEWINSTANCEMASK_INVALIDARGS
D3D12_MESSAGE_ID_VIEW_INSTANCING_UNSUPPORTED
D3D12_MESSAGE_ID_VIEW_INSTANCING_INVALIDARGS
D3D12_MESSAGE_ID_COPYTEXTUREREGION_MISMATCH_DE_CODE_REFERENCE_ONLY_FLAG
D3D12_MESSAGE_ID_COPYRESOURCE_MISMATCH_DECODE_REFERENCE_ONLY_FLAG
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_FAILURE
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODE_HEAP_CAPS_UNSUPPORTED
D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_INVALID_IN_PUT
D3D12_MESSAGE_ID_CREATE_VIDEO_DECODER_UNSUPPORTED
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_META_DATA_ERROR
D3D12_MESSAGE_ID_CREATEGRAPHICSPipelineState_VIEW_INSTANCING_VERTEX_SIZE_EXCEEDED

D3D12_MESSAGE_ID_CREATEGRAPHICPIPELINESTATE_RUNTIME_INTERNAL_ERROR
D3D12_MESSAGE_ID_NO_VIDEO_API_SUPPORT
D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_INVALID_INPUT
D3D12_MESSAGE_ID_CREATE_VIDEO_PROCESSOR_CAPS_FAILURE
D3D12_MESSAGE_ID_VIDEO_PROCESS_SUPPORT_UNSUPPORTED_FORMAT
D3D12_MESSAGE_ID_VIDEO_DECODE_FRAME_INVALID_ARGUMENT
D3D12_MESSAGE_ID_ENQUEUE_MAKE_RESIDENT_INVALID_FLAGS
D3D12_MESSAGE_ID_OPENEXISTINGHEAP_UNSUPPORTED
D3D12_MESSAGE_ID_VIDEO_PROCESS_FRAMES_INVALID_ARGUMENT
D3D12_MESSAGE_ID_VIDEO_DECODE_SUPPORT_UNSUPPORTED
D3D12_MESSAGE_ID_CREATE_COMMANDRECORDER
D3D12_MESSAGE_ID_LIVE_COMMANDRECORDER
D3D12_MESSAGE_ID_DESTROY_COMMANDRECORDER
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_VIDEO_NOT_SUPPORTED
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_SUPPORT_FLAGS
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_INVALID_FLAGS
D3D12_MESSAGE_ID_CREATE_COMMAND_RECORDER_MORE_RECORDERS_THAN_LOGICAL_PROCESSORS
D3D12_MESSAGE_ID_CREATE_COMMANDPOOL
D3D12_MESSAGE_ID_LIVE_COMMANDPOOL
D3D12_MESSAGE_ID_DESTROY_COMMANDPOOL

D3D12_MESSAGE_ID_CREATE_COMMAND_POOL_INVALID_FLAGS
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_VIDEO_NOT_SUPPORTED
D3D12_MESSAGE_ID_COMMAND_RECORDER_SUPPORT_FLAGS_MISMATCH
D3D12_MESSAGE_ID_COMMAND_RECORDER_CONTENTION
D3D12_MESSAGE_ID_COMMAND_RECORDER_USAGE_WITH_CREATECOMMANDLIST_COMMAND_LIST
D3D12_MESSAGE_ID_COMMAND_ALLOCATOR_USAGE_WITH_CREATECOMMANDLIST1_COMMAND_LIST
D3D12_MESSAGE_ID_CANNOT_EXECUTE_EMPTY_COMMAND_LIST
D3D12_MESSAGE_ID_CANNOT_RESET_COMMAND_POOL_WITH_OPEN_COMMAND_LISTS
D3D12_MESSAGE_ID_CANNOT_USE_COMMAND_RECORDER_WITHOUT_CURRENT_TARGET
D3D12_MESSAGE_ID_CANNOT_CHANGE_COMMAND_RECORDER_TARGET_WHILE_RECORDING
D3D12_MESSAGE_ID_COMMAND_POOL_SYNC
D3D12_MESSAGE_ID_EVICT_UNDERFLOW
D3D12_MESSAGE_ID_CREATE_META_COMMAND
D3D12_MESSAGE_ID_LIVE_META_COMMAND
D3D12_MESSAGE_ID_DESTROY_META_COMMAND
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_DST_RESOURCE
D3D12_MESSAGE_ID_COPYBUFFERREGION_INVALID_SRC_RESOURCE
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_DST_RESOURCE
D3D12_MESSAGE_ID_ATOMICCOPYBUFFER_INVALID_SRC_RESOURCE
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_BUFFER

D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_NULL_RESOURCE_DESC
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_UNSUPPORTED
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_DIMENSION
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_FLAGS
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_BUFFER_OFFSET
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_DIMENSION
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_INVALID_RESOURCE_FLAGS
D3D12_MESSAGE_ID_CREATEPLACEDRESOURCEONBUFFER_OUTOFMEMORY_RETURN
D3D12_MESSAGE_ID_CANNOT_CREATE_GRAPHICS_AND_VIDEO_COMMAND_RECORDER
D3D12_MESSAGE_ID_UPDATETILEMAPPINGS_POSSIBLY_MISMATCHING_PROPERTIES
D3D12_MESSAGE_ID_CREATE_COMMAND_LIST_INVALID_COMMAND_LIST_TYPE
D3D12_MESSAGE_ID_CLEARUNORDEREDACCESSVIEW_INCOMPATIBLE_WITH_STRUCTURED_BUFFERS
D3D12_MESSAGE_ID_COMPUTE_ONLY_DEVICE_OPERATION_UNSUPPORTED
D3D12_MESSAGE_ID_BUILD_RAYTRACING_ACCELERATION_STRUCTURE_INVALID
D3D12_MESSAGE_ID_EMIT_RAYTRACING_ACCELERATION_STRUCTURE_POSTBUILD_INFO_INVALID
D3D12_MESSAGE_ID_COPY_RAYTRACING_ACCELERATION_STRUCTURE_INVALID
D3D12_MESSAGE_ID_DISPATCH_RAYS_INVALID
D3D12_MESSAGE_ID_GET_RAYTRACING_ACCELERATION_STRUCTURE_PREBUILD_INFO_INVALID
D3D12_MESSAGE_ID_CREATE_LIFETIMETRACKER

D3D12_MESSAGE_ID_LIVE_LIFETIMETRACKER
D3D12_MESSAGE_ID_DESTROY_LIFETIMETRACKER
D3D12_MESSAGE_ID_DESTROYOWNEDOBJECT_OBJECTNOTOWNED
D3D12_MESSAGE_ID_CREATE_TRACKEDWORKLOAD
D3D12_MESSAGE_ID_LIVE_TRACKEDWORKLOAD
D3D12_MESSAGE_ID_DESTROY_TRACKEDWORKLOAD
D3D12_MESSAGE_ID_RENDER_PASS_ERROR
D3D12_MESSAGE_ID_META_COMMAND_ID_INVALID
D3D12_MESSAGE_ID_META_COMMAND_UNSUPPORTED_PARAMETERS
D3D12_MESSAGE_ID_META_COMMAND_FAILED_ENUMERATION
D3D12_MESSAGE_ID_META_COMMAND_PARAMETER_SIZE_MISMATCH
D3D12_MESSAGE_ID_UNINITIALIZED_META_COMMAND
D3D12_MESSAGE_ID_META_COMMAND_INVALID_GPU_VIRTUAL_ADDRESS
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDLIST
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDLIST
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDLIST
D3D12_MESSAGE_ID_CREATE_VIDEOENCODECOMMANDQUEUE
D3D12_MESSAGE_ID_LIVE_VIDEOENCODECOMMANDQUEUE
D3D12_MESSAGE_ID_DESTROY_VIDEOENCODECOMMANDQUEUE
D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONESTIMATOR
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONESTIMATOR
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONESTIMATOR

D3D12_MESSAGE_ID_CREATE_VIDEOMOTIONVECTORHEAP
D3D12_MESSAGE_ID_LIVE_VIDEOMOTIONVECTORHEAP
D3D12_MESSAGE_ID_DESTROY_VIDEOMOTIONVECTORHEAP
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOADS
D3D12_MESSAGE_ID_MULTIPLE_TRACKED_WORKLOAD_PAIRS
D3D12_MESSAGE_ID_OUT_OF_ORDER_TRACKED_WORKLOAD_PAIR
D3D12_MESSAGE_ID_CANNOT_ADD_TRACKED_WORKLOAD
D3D12_MESSAGE_ID_INCOMPLETE_TRACKED_WORKLOAD_PAIR
D3D12_MESSAGE_ID_CREATE_STATE_OBJECT_ERROR
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_ERROR
D3D12_MESSAGE_ID_GET_SHADER_STACK_SIZE_ERROR
D3D12_MESSAGE_ID_GET_PIPELINE_STACK_SIZE_ERROR
D3D12_MESSAGE_ID_SET_PIPELINE_STACK_SIZE_ERROR
D3D12_MESSAGE_ID_GET_SHADER_IDENTIFIER_SIZE_INVALID
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_INVALID
D3D12_MESSAGE_ID_CHECK_DRIVER_MATCHING_IDENTIFIER_DRIVER_REPORTED_ISSUE
D3D12_MESSAGE_ID_RENDER_PASS_INVALID_RESOURCE_BARRIER
D3D12_MESSAGE_ID_RENDER_PASS_DISALLOWED_API_CALLED
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_NEST_RENDER_PASSES
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_END_WITHOUT_BEGIN
D3D12_MESSAGE_ID_RENDER_PASS_CANNOT_CLOSE_COMMAND_LIST

D3D12_MESSAGE_ID_RENDER_PASS_GPU_WORK_WHILE_SUSPENDED	
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_SUSPEND_RESUME	
D3D12_MESSAGE_ID_RENDER_PASS_NO_PRIOR_SUSPEND_WITHIN_EXECUTECOMMANDLISTS	
D3D12_MESSAGE_ID_RENDER_PASS_NO_SUBSEQUENT_RESUME_WITHIN_EXECUTECOMMANDLISTS	
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_COMMAND_QUEUE_MISMATCH	
D3D12_MESSAGE_ID_TRACKED_WORKLOAD_NOT_SUPPORTED	
D3D12_MESSAGE_ID_RENDER_PASS_MISMATCHING_NO_ACROSS	
D3D12_MESSAGE_ID_RENDER_PASS_UNSUPPORTED_RESOLVE	
D3D12_MESSAGE_ID_D3D12_MESSAGES_END	

Remarks

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

Requirements

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

D3D12_MESSAGE_SEVERITY enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Debug message severity levels for an information queue.

Syntax

```
typedef enum D3D12_MESSAGE_SEVERITY {
    D3D12_MESSAGE_SEVERITY_CORRUPTION,
    D3D12_MESSAGE_SEVERITY_ERROR,
    D3D12_MESSAGE_SEVERITY_WARNING,
    D3D12_MESSAGE_SEVERITY_INFO,
    D3D12_MESSAGE_SEVERITY_MESSAGE
} ;
```

Constants

D3D12_MESSAGE_SEVERITY_CORRUPTION	Indicates a corruption error.
D3D12_MESSAGE_SEVERITY_ERROR	Indicates an error.
D3D12_MESSAGE_SEVERITY_WARNING	Indicates a warning.
D3D12_MESSAGE_SEVERITY_INFO	Indicates an information message.
D3D12_MESSAGE_SEVERITY_MESSAGE	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

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

D3D12_RLDO_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies options for the amount of information to report about a live device object's lifetime.

Syntax

```
typedef enum D3D12_RLDO_FLAGS {  
    D3D12_RLDO_NONE,  
    D3D12_RLDO_SUMMARY,  
    D3D12_RLDO_DETAIL,  
    D3D12_RLDO_IGNORE_INTERNAL  
} ;
```

Constants

D3D12_RLDO_NONE	
D3D12_RLDO_SUMMARY	Obtain a summary about a live device object's lifetime.
D3D12_RLDO_DETAIL	Obtain detailed information about a live device object's lifetime.
D3D12_RLDO_IGNORE_INTERNAL	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

Header	d3d12sdklayers.h
--------	------------------

See also

[Debug Layer Enumerations](#)

ID3D12Debug interface

4/29/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12Debug** interface has these methods.

METHOD	DESCRIPTION
ID3D12Debug::EnableDebugLayer	Enables the debug layer.

Remarks

This interface is obtained by querying it from [D3D12GetDebugInterface](#).

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

ID3D12Debug::EnableDebugLayer method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Enables the debug layer.

Syntax

```
void EnableDebugLayer();
```

Parameters

This method has no parameters.

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.

```
// 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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug](#)

ID3D12Debug1 interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12Debug1** interface has these methods.

METHOD	DESCRIPTION
ID3D12Debug1::EnableDebugLayer	Enables the debug layer.
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

ID3D12Debug1::EnableDebugLayer method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Enables the debug layer.

Syntax

```
void EnableDebugLayer();
```

Parameters

This method has no parameters.

Return value

None

Remarks

This method is identical to [ID3D12Debug::EnableDebugLayer](#).

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

ID3D12Debug1::SetEnableGPUBasedValidation method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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.

For more information, see [Using D3D12 Debug Layer GPU-Based Validation](#).

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

ID3D12Debug1::SetEnableSynchronizedCommandQueueValidation method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug1](#)

ID3D12Debug2 interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

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

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces IUnknown](#)

ID3D12Debug2::SetGPUBasedValidationFlags method

4/22/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug2](#)

ID3D12Debug3 interface

6/25/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
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.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces IUnknown](#)

ID3D12Debug3::SetEnableGPUBasedValidation method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

ID3D12Debug3::SetEnableSynchronizedCommandQueueValidation method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

ID3D12Debug3::SetGPUBasedValidationFlags method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12Debug3](#)

ID3D12DebugCommandList interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12DebugCommandList** interface has these methods.

METHOD	DESCRIPTION
ID3D12DebugCommandList::AssertResourceState	Checks whether a resource, or subresource, is in a specified state, or not.
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

ID3D12DebugCommandList::AssertResourceState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Checks whether a resource, or subresource, is in a specified state, or not.

Syntax

```
BOOL AssertResourceState(  
    ID3D12Resource *pResource,  
    UINT             Subresource,  
    UINT             State  
>;
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

ID3D12DebugCommandList

ID3D12DebugCommandList::GetFeatureMask method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Returns the debug feature flags that have been set on a command list.

Syntax

```
D3D12_DEBUG_FEATURE GetFeatureMask();
```

Parameters

This method has no parameters.

Return value

Type: [D3D12_DEBUG_FEATURE](#)

A bit mask containing the set debug features.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList](#)

ID3D12DebugCommandList::SetFeatureMask method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList](#)

ID3D12DebugCommandList1 interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12DebugCommandList1** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

ID3D12DebugCommandList1::AssertResourceState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
BOOL AssertResourceState(  
    ID3D12Resource *pResource,  
    UINT           Subresource,  
    UINT           State  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList1](#)

ID3D12DebugCommandList1::GetDebugParameter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets optional Command List Debug Layer settings.

Syntax

```
HRESULT GetDebugParameter(  
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE Type,  
    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*.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandList1](#)

[SetDebugParameter](#)

ID3D12DebugCommandList1::SetDebugParameter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Modifies optional Debug Layer settings of a command list.

Syntax

```
HRESULT SetDebugParameter(  
    D3D12_DEBUG_COMMAND_LIST_PARAMETER_TYPE Type,  
    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.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetDebugParameter](#)

[ID3D12DebugCommandList1](#)

ID3D12DebugCommandQueue interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The ID3D12DebugCommandQueue interface has these methods.

METHOD	DESCRIPTION
ID3D12DebugCommandQueue::AssertResourceState	Checks whether a resource, or subresource, is in a specified state, or not.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

ID3D12DebugCommandQueue::AssertResourceState method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Checks whether a resource, or subresource, is in a specified state, or not.

Syntax

```
BOOL AssertResourceState(  
    ID3D12Resource *pResource,  
    UINT             Subresource,  
    UINT             State  
) ;
```

Parameters

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 [ID3D13CommandQueue::CopyTileMappings](#)).

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugCommandQueue](#)

ID3D12DebugDevice interface

6/25/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12DebugDevice** interface has these methods.

METHOD	DESCRIPTION
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.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

ID3D12DebugDevice::GetFeatureMask method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a bit field of flags that indicates which debug features are on or off.

Syntax

```
D3D12_DEBUG_FEATURE GetFeatureMask();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice](#)

[ID3D12DebugDevice::SetFeatureMask](#)

ID3D12DebugDevice::ReportLiveDeviceObjects method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Reports information about a device object's lifetime.

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice](#)

ID3D12DebugDevice::SetFeatureMask method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetFeatureMask](#)

[ID3D12DebugDevice](#)

ID3D12DebugDevice1 interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

Specifies device-wide debug layer settings.

Inheritance

The **ID3D12DebugDevice1** interface inherits from the [IUnknown](#) interface. **ID3D12DebugDevice1** also has these types of members:

- [Methods](#)

Methods

The **ID3D12DebugDevice1** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

[Using D3D12 Debug Layer GPU-Based Validation](#)

ID3D12DebugDevice1::GetDebugParameter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets optional device-wide Debug Layer settings.

Syntax

```
HRESULT GetDebugParameter(  
    D3D12_DEBUG_DEVICE_PARAMETER_TYPE Type,  
    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.

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

Requirement	Description
Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice1](#)

[SetDebugParameter](#)

ID3D12DebugDevice1::ReportLiveDeviceObjects method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12DebugDevice1](#)

ID3D12DebugDevice1::SetDebugParameter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Modifies the D3D12 optional device-wide Debug Layer settings.

Syntax

```
HRESULT SetDebugParameter(  
    D3D12_DEBUG_DEVICE_PARAMETER_TYPE Type,  
    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.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[GetDebugParameter](#)

[ID3D12DebugDevice1](#)

ID3D12InfoQueue interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12InfoQueue** interface has these methods.

METHOD	DESCRIPTION
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::AddStorageFilterEntries	Add storage filters to the top of the storage-filter stack.
ID3D12InfoQueue::ClearRetrievalFilter	Remove a retrieval filter from the top of the retrieval-filter stack.
ID3D12InfoQueue::ClearStorageFilter	Remove a storage filter from the top of the storage-filter stack.
ID3D12InfoQueue::ClearStoredMessages	Clear all messages from the message queue.
ID3D12InfoQueue::GetBreakOnCategory	Get a message category to break on when a message with that category passes through the storage filter.
ID3D12InfoQueue::GetBreakOnID	Get a message identifier to break on when a message with that identifier passes through the storage filter.
ID3D12InfoQueue::GetBreakOnSeverity	Get a message severity level to break on when a message with that severity level passes through the storage filter.
ID3D12InfoQueue::GetMessage	Get a message from the message queue.
ID3D12InfoQueue::GetMessageCountLimit	Get the maximum number of messages that can be added to the message queue.

METHOD	DESCRIPTION
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::GetNumMessagesDeniedByStorageFilter	Get the number of messages that were denied passage through a storage filter.
ID3D12InfoQueue::GetNumMessagesDiscardedByMessageCountLimit	Get the number of messages that were discarded due to the message count limit.
ID3D12InfoQueue::GetNumStoredMessages	Get the number of messages currently stored in the message queue.
ID3D12InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter	Get the number of messages that are able to pass through a retrieval filter.
ID3D12InfoQueue::GetRetrievalFilter	Get the retrieval filter at the top of the retrieval-filter stack.
ID3D12InfoQueue::GetRetrievalFilterStackSize	Get the size of the retrieval-filter stack in bytes.
ID3D12InfoQueue::GetStorageFilter	Get the storage filter at the top of the storage-filter stack.
ID3D12InfoQueue::GetStorageFilterStackSize	Get the size of the storage-filter stack in bytes.
ID3D12InfoQueue::PopRetrievalFilter	Pop a retrieval filter from the top of the retrieval-filter stack.
ID3D12InfoQueue::PopStorageFilter	Pop a storage filter from the top of the storage-filter stack.
ID3D12InfoQueue::PushCopyOfRetrievalFilter	Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack.
ID3D12InfoQueue::PushCopyOfStorageFilter	Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack.
ID3D12InfoQueue::PushEmptyRetrievalFilter	Push an empty retrieval filter onto the retrieval-filter stack.
ID3D12InfoQueue::PushEmptyStorageFilter	Push an empty storage filter onto the storage-filter stack.
ID3D12InfoQueue::PushRetrievalFilter	Push a retrieval filter onto the retrieval-filter stack.
ID3D12InfoQueue::PushStorageFilter	Push a storage filter onto the storage-filter stack.
ID3D12InfoQueue::SetBreakOnCategory	Set a message category to break on when a message with that category passes through the storage filter.
ID3D12InfoQueue::SetBreakOnID	Set a message identifier to break on when a message with that identifier passes through the storage filter.
ID3D12InfoQueue::SetBreakOnSeverity	Set a message severity level to break on when a message with that severity level passes through the storage filter.

METHOD	DESCRIPTION
ID3D12InfoQueue::SetMessageCountLimit	Set the maximum number of messages that can be added to the message queue.
ID3D12InfoQueue::SetMuteDebugOutput	Set a boolean that turns the debug output on or off.

Remarks

This interface is obtained by querying it from the [ID3D12Device](#) using `IUnknown::QueryInterface`.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[Debug Layer Interfaces](#)

[IUnknown](#)

ID3D12InfoQueue::AddApplicationMessage method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT AddApplicationMessage(  
    D3D12_MESSAGE_SEVERITY Severity,  
    LPCSTR                 pDescription  
) ;
```

Parameters

Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

Severity of a message.

pDescription

Type: [LPCSTR](#)

Specifies the message string.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::AddMessage method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT AddMessage(  
    D3D12_MESSAGE_CATEGORY Category,  
    D3D12_MESSAGE_SEVERITY Severity,  
    D3D12_MESSAGE_ID        ID,  
    LPCSTR                  pDescription  
) ;
```

Parameters

Category

Type: [D3D12_MESSAGE_CATEGORY](#)

Category of a message.

Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

Severity of a message.

ID

Type: [D3D12_MESSAGE_ID](#)

Unique identifier of a message.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::AddRetrievalFilterEntries method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Add storage filters to the top of the retrieval-filter stack.

Syntax

```
HRESULT AddRetrievalFilterEntries(  
    D3D12_INFO_QUEUE_FILTER *pFilter  
)
```

Parameters

`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:

```
D3D12_MESSAGE_CATEGORY cats[] = { ..., ..., ... };  
D3D12_MESSAGE_SEVERITY sevs[] = { ..., ..., ... };  
UINT 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 = sizeof(cats / sizeof(D3D12_MESSAGE_CATEGORY));  
filter.AllowList.pCategoryList = cats;  
filter.AllowList.NumSeverities = sizeof(sevs / sizeof(D3D12_MESSAGE_SEVERITY));  
filter.AllowList.pSeverityList = sevs;  
filter.AllowList.NumIDs = sizeof(ids) / sizeof(UINT);  
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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::AddStorageFilterEntries method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Add storage filters to the top of the storage-filter stack.

Syntax

```
HRESULT AddStorageFilterEntries(  
    D3D12_INFO_QUEUE_FILTER *pFilter  
>);
```

Parameters

`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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::ClearRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void ClearRetrievalFilter();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::ClearStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void ClearStorageFilter();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::ClearStoredMessages method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Clear all messages from the message queue.

Syntax

```
void ClearStoredMessages();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetBreakOnCategory method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get a message category to break on when a message with that category passes through the storage filter.

Syntax

```
BOOL GetBreakOnCategory(  
    D3D12_MESSAGE_CATEGORY Category  
>);
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetBreakOnID method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get a message identifier to break on when a message with that identifier passes through the storage filter.

Syntax

```
BOOL GetBreakOnID(  
    D3D12_MESSAGE_ID ID  
>);
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetBreakOnSeverity method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get a message severity level to break on when a message with that severity level passes through the storage filter.

Syntax

```
BOOL GetBreakOnSeverity(  
    D3D12_MESSAGE_SEVERITY Severity  
>);
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetMessage method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get a message from the message queue.

Syntax

```
HRESULT GetMessage(  
    UINT64        MessageIndex,  
    D3D12_MESSAGE *pMessage,  
    SIZE_T        *pMessageByteLength  
) ;
```

Parameters

`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.

`pMessage`

Type: `D3D12_MESSAGE*`

Returned message.

`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:

```
// 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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetMessageCountLimit method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetMessageCountLimit();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetMuteDebugOutput method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
BOOL GetMuteDebugOutput();
```

Parameters

This method has no parameters.

Return value

Type: **BOOL**

Whether the debug output is on or off (true for on, false for off).

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetNumMessagesAllowedByStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetNumMessagesAllowedByStorageFilter();
```

Parameters

This method has no parameters.

Return value

Type: **UINT64**

Number of messages allowed by a storage filter.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetNumMessagesDeniedByStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetNumMessagesDeniedByStorageFilter();
```

Parameters

This method has no parameters.

Return value

Type: **UINT64**

Number of messages denied by a storage filter.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetNumMessagesDiscardedByMessageCountLimit method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetNumMessagesDiscardedByMessageCountLimit();
```

Parameters

This method has no parameters.

Return value

Type: **UINT64**

Number of messages discarded.

Remarks

Get and set the message count limit with [GetMessageCountLimit](#) and [SetMessageCountLimit](#), respectively.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetNumStoredMessages method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT64 GetNumStoredMessages();
```

Parameters

This method has no parameters.

Return value

Type: **UINT64**

Number of messages currently stored in the message queue.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetNumStoredMessagesAllowedByRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get the number of messages that are able to pass through a retrieval filter.

Syntax

```
UINT64 GetNumStoredMessagesAllowedByRetrievalFilter();
```

Parameters

This method has no parameters.

Return value

Type: **UINT64**

Number of messages allowed by a retrieval filter.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get the retrieval filter at the top of the retrieval-filter stack.

Syntax

```
HRESULT GetRetrievalFilter(  
    D3D12_INFO_QUEUE_FILTER *pFilter,  
    SIZE_T                 *pFilterByteLength  
) ;
```

Parameters

`pFilter`

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Retrieval filter at the top of the retrieval-filter stack.

`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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetRetrievalFilterStackSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get the size of the retrieval-filter stack in bytes.

Syntax

```
UINT GetRetrievalFilterStackSize();
```

Parameters

This method has no parameters.

Return value

Type: **UINT**

Size of the retrieval-filter stack in bytes.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get the storage filter at the top of the storage-filter stack.

Syntax

```
HRESULT GetStorageFilter(  
    D3D12_INFO_QUEUE_FILTER *pFilter,  
    SIZE_T                 *pFilterByteLength  
) ;
```

Parameters

pFilter

Type: [D3D12_INFO_QUEUE_FILTER*](#)

Storage filter at the top of the storage-filter stack.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::GetStorageFilterStackSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Get the size of the storage-filter stack in bytes.

Syntax

```
UINT GetStorageFilterStackSize();
```

Parameters

This method has no parameters.

Return value

Type: **UINT**

Size of the storage-filter stack in bytes.

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PopRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void PopRetrievalFilter();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PopStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
void PopStorageFilter();
```

Parameters

This method has no parameters.

Return value

None

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushCopyOfRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push a copy of retrieval filter currently on the top of the retrieval-filter stack onto the retrieval-filter stack.

Syntax

```
HRESULT PushCopyOfRetrievalFilter();
```

Parameters

This method has no parameters.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushCopyOfStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push a copy of storage filter currently on the top of the storage-filter stack onto the storage-filter stack.

Syntax

```
HRESULT PushCopyOfStorageFilter();
```

Parameters

This method has no parameters.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushEmptyRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push an empty retrieval filter onto the retrieval-filter stack.

Syntax

```
HRESULT PushEmptyRetrievalFilter();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushEmptyStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push an empty storage filter onto the storage-filter stack.

Syntax

```
HRESULT PushEmptyStorageFilter();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushRetrievalFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push a retrieval filter onto the retrieval-filter stack.

Syntax

```
HRESULT PushRetrievalFilter(  
    D3D12_INFO_QUEUE_FILTER *pFilter  
>);
```

Parameters

`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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::PushStorageFilter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Push a storage filter onto the storage-filter stack.

Syntax

```
HRESULT PushStorageFilter(  
    D3D12_INFO_QUEUE_FILTER *pFilter  
>);
```

Parameters

`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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::SetBreakOnCategory method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT SetBreakOnCategory(  
    D3D12_MESSAGE_CATEGORY Category,  
    BOOL                 bEnable  
)
```

Parameters

Category

Type: [D3D12_MESSAGE_CATEGORY](#)

Message category to break on.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::SetBreakOnID method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT SetBreakOnID(  
    D3D12_MESSAGE_ID ID,  
    BOOL             bEnable  
)
```

Parameters

ID

Type: [D3D12_MESSAGE_ID](#)

Message identifier to break on.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::SetBreakOnSeverity method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Set a message severity level to break on when a message with that severity level passes through the storage filter.

Syntax

```
HRESULT SetBreakOnSeverity(  
    D3D12_MESSAGE_SEVERITY Severity,  
    BOOL                bEnable  
)
```

Parameters

Severity

Type: [D3D12_MESSAGE_SEVERITY](#)

A message severity level to break on.

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::SetMessageCountLimit method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT SetMessageCountLimit(  
    UINT64 MessageCountLimit  
>);
```

Parameters

`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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12InfoQueue::SetMuteDebugOutput method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Set a boolean that turns the debug output on or off.

Syntax

```
void SetMuteDebugOutput(  
    BOOL bMute  
>);
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h

See also

[ID3D12InfoQueue](#)

ID3D12SharingContract interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12SharingContract** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12sdklayers.h (include D3D12.h)

See also

[Core interfaces](#), [IUnknown](#)

ID3D12SharingContract::Present method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

```
void Present(
    ID3D12Resource *pResource,
    UINT             Subresource,
    HWND             window
);
```

Parameters

`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

Target Platform	Windows
Header	d3d12sdklayers.h (include D3D12.h)

See also

[ID3D12SharingContract](#)

ID3D12SharingContract::SharedFenceSignal method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Signals a shared fence between the D3D layers and diagnostics tools.

Syntax

```
void SharedFenceSignal(  
    ID3D12Fence *pFence,  
    UINT64      FenceValue  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12sdklayers.h (include D3D12.h)

See also

[ID3D12SharingContract](#)

d3d12shader.h header

2/7/2020 • 2 minutes to read • [Edit Online](#)

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

- [Direct3D 12 Graphics](#) d3d12shader.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
ID3D12FunctionParameterReflection	A function-parameter-reflection interface accesses function-parameter info.
ID3D12FunctionReflection	A function-reflection interface accesses function info.
ID3D12LibraryReflection	A library-reflection interface accesses library info.
ID3D12ShaderReflection	A shader-reflection interface accesses shader information.
ID3D12ShaderReflectionConstantBuffer	This shader-reflection interface provides access to a constant buffer.
ID3D12ShaderReflectionType	This shader-reflection interface provides access to variable type.
ID3D12ShaderReflectionVariable	This shader-reflection interface provides access to a variable.

Structures

TITLE	DESCRIPTION
D3D12_FUNCTION_DESC	Describes a function.
D3D12_LIBRARY_DESC	Describes a library.
D3D12_PARAMETER_DESC	Describes a function parameter.
D3D12_SHADER_BUFFER_DESC	Describes a shader constant-buffer.
D3D12_SHADER_DESC	Describes a shader.
D3D12_SHADER_INPUT_BIND_DESC	Describes how a shader resource is bound to a shader input.
D3D12_SHADER_TYPE_DESC	Describes a shader-variable type.
D3D12_SHADER_VARIABLE_DESC	Describes a shader variable.
D3D12_SIGNATURE_PARAMETER_DESC	Describes a shader signature.

Enumerations

TITLE	DESCRIPTION
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_FUNCTION_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a function.

Syntax

```
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

Header	
	d3d12shader.h

See also

[ID3D12FunctionReflection::GetDesc](#)

[Shader Structures](#)

D3D12_LIBRARY_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a library.

Syntax

```
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

Header	d3d12shader.h

See also

[ID3D12LibraryReflection::GetDesc](#)

[Shader Structures](#)

D3D12_PARAMETER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a function parameter.

Syntax

```
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

<code>Header</code>	<code>d3d12shader.h</code>

See also

[ID3D12FunctionParameterReflection::GetDesc](#)

[Shader Structures](#)

D3D12_SHADER_BUFFER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader constant-buffer.

Syntax

```
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

Header	d3d12shader.h
--------	---------------

See also

[Shader Structures](#)

D3D12_SHADER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader.

Syntax

```
typedef struct _D3D12_SHADER_DESC {
    UINT             Version;
    LPCSTR           Creator;
    UINT             Flags;
    UINT             ConstantBuffers;
    UINT             BoundResources;
    UINT             InputParameters;
    UINT             OutputParameters;
    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             CutInstructionCount;
    UINT             EmitInstructionCount;
    D3D_PRIMITIVE_TOPOLOGY   GSOutputTopology;
    UINT             GSMaxOutputVertexCount;
    D3D_PRIMITIVE      InputPrimitive;
    UINT             PatchConstantParameters;
    UINT             cGSInstanceCount;
    UINT             cControlPoints;
    D3D_TESSELLATOR_OUTPUT_PRIMITIVE HSOutputPrimitive;
    D3D_TESSELLATOR_PARTITIONING   HSPartitioning;
    D3D_TESSELLATOR_DOMAIN       TessellatorDomain;
    UINT             cBarrierInstructions;
    UINT             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.

`cGSIInstanceCount`

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.

`HSPartitioning`

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

Header	
	d3d12shader.h

See also

[Shader Structures](#)

D3D12_SHADER_INPUT_BIND_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes how a shader resource is bound to a shader input.

Syntax

```
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.

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

Header	d3d12shader.h

See also

[Shader Structures](#)

D3D12_SHADER_TYPE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader-variable type.

Syntax

```
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

Header	
	d3d12shader.h

See also

[Shader Structures](#)

D3D12_SHADER_VARIABLE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader variable.

Syntax

```
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

Header	
	d3d12shader.h

See also

[Shader Structures](#)

D3D12_SHADER_VERSION_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Enumerates the types of shaders that Direct3D recognizes.

Used to encode the **Version** member of the [D3D12_SHADER_DESC](#) structure.

Syntax

```
typedef enum D3D12_SHADER_VERSION_TYPE {  
    D3D12_SHVER_PIXEL_SHADER,  
    D3D12_SHVER_VERTEX_SHADER,  
    D3D12_SHVER_GEOMETRY_SHADER,  
    D3D12_SHVER_HULL_SHADER,  
    D3D12_SHVER_DOMAIN_SHADER,  
    D3D12_SHVER_COMPUTE_SHADER,  
    D3D12_SHVER_RESERVED0  
} ;
```

Constants

D3D12_SHVER_PIXEL_SHADER	Pixel shader.
D3D12_SHVER_VERTEX_SHADER	Vertex shader.
D3D12_SHVER_GEOMETRY_SHADER	Geometry shader.
D3D12_SHVER_HULL_SHADER	Hull shader.
D3D12_SHVER_DOMAIN_SHADER	Domain shader.
D3D12_SHVER_COMPUTE_SHADER	Compute shader.
D3D12_SHVER_RESERVED0	Indicates the end of the enumeration.

Requirements

Header	d3d12shader.h
--------	---------------

See also

[Shader Enumerations](#)

D3D12_SIGNATURE_PARAMETER_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a shader signature.

Syntax

```
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

Header	
	d3d12shader.h

See also

[Shader Structures](#)

ID3D12FunctionParameterReflection interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12FunctionParameterReflection::GetDesc	Fills the parameter descriptor structure for the function's parameter.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

ID3D12FunctionParameterReflection::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT GetDesc(  
    D3D12_PARAMETER_DESC *pDesc  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionParameterReflection](#)

ID3D12FunctionReflection interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
ID3D12FunctionReflection::GetConstantBufferByIndex	Gets a constant buffer by index for a function.
ID3D12FunctionReflection::GetConstantBufferByName	Gets a constant buffer by name for a function.
ID3D12FunctionReflection::GetDesc	Fills the function descriptor structure for the function.
ID3D12FunctionReflection::GetFunctionParameter	Gets the function parameter reflector.
ID3D12FunctionReflection::GetResourceBindingDesc	Gets a description of how a resource is bound to a function.
ID3D12FunctionReflection::GetResourceBindingDescByName	Gets a description of how a resource is bound to a function.
ID3D12FunctionReflection::GetVariableByName	Gets a variable by name.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

ID3D12FunctionReflection::GetConstantBufferByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets a constant buffer by index for a function.

Syntax

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    UINT BufferIndex  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetConstantBufferByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a constant buffer by name for a function.

Syntax

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    LPCSTR Name  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Fills the function descriptor structure for the function.

Syntax

```
HRESULT GetDesc(  
    D3D12_FUNCTION_DESC *pDesc  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetFunctionParameter method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the function parameter reflector.

Syntax

```
ID3D12FunctionParameterReflection * GetFunctionParameter(  
    INT ParameterIndex  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetResourceBindingDesc method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets a description of how a resource is bound to a function.

Syntax

```
HRESULT GetResourceBindingDesc(  
    UINT             ResourceIndex,  
    D3D12_SHADER_INPUT_BIND_DESC *pDesc  
)
```

Parameters

ResourceIndex

Type: [UINT](#)

A zero-based resource index.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetResourceBindingDescByName method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets a description of how a resource is bound to a function.

Syntax

```
HRESULT GetResourceBindingDescByName(  
    LPCSTR             Name,  
    D3D12_SHADER_INPUT_BIND_DESC *pDesc  
)
```

Parameters

Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12FunctionReflection::GetVariableByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a variable by name.

Syntax

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    LPCSTR Name  
) ;
```

Parameters

Name

Type: [LPCSTR](#)

A pointer to a string containing the variable name.

Return value

Type: [ID3D12ShaderReflectionVariable*](#)

Returns a [ID3D12ShaderReflectionVariable Interface](#) interface.

Requirements

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12FunctionReflection](#)

ID3D12LibraryReflection interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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](#)

Methods

The **ID3D12LibraryReflection** interface has these methods.

METHOD	DESCRIPTION
ID3D12LibraryReflection::GetDesc	Fills the library descriptor structure for the library reflection.
ID3D12LibraryReflection::GetFunctionByIndex	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

Target Platform	Windows
Header	d3d12shader.h

See also

[IUnknown](#)

[Shader Interfaces](#)

ID3D12LibraryReflection::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Fills the library descriptor structure for the library reflection.

Syntax

```
HRESULT GetDesc(  
    D3D12_LIBRARY_DESC *pDesc  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12LibraryReflection](#)

ID3D12LibraryReflection::GetFunctionByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets the function reflector.

Syntax

```
ID3D12FunctionReflection * GetFunctionByIndex(  
    INT FunctionIndex  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12LibraryReflection](#)

ID3D12ShaderReflection interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

A shader-reflection interface accesses shader information.

Inheritance

The **ID3D12ShaderReflection** interface inherits from the [IUnknown](#) interface. **ID3D12ShaderReflection** also has these types of members:

- [Methods](#)

Methods

The **ID3D12ShaderReflection** interface has these methods.

METHOD	DESCRIPTION
ID3D12ShaderReflection::GetBitwiseInstructionCount	Gets the number of bitwise instructions.
ID3D12ShaderReflection::GetConstantBufferByIndex	Gets a constant buffer by index.
ID3D12ShaderReflection::GetConstantBufferByName	Gets a constant buffer by name.
ID3D12ShaderReflection::GetConversionInstructionCount	Gets the number of conversion instructions.
ID3D12ShaderReflection::GetDesc	Gets a shader description.
ID3D12ShaderReflection::GetGSIInputPrimitive	Gets the geometry-shader input-primitive description.
ID3D12ShaderReflection::GetInputParameterDesc	Gets an input-parameter description for a shader.
ID3D12ShaderReflection::GetMinFeatureLevel	Gets the minimum feature level.
ID3D12ShaderReflection::GetMovcInstructionCount	Gets the number of Movc instructions.
ID3D12ShaderReflection::GetMovInstructionCount	Gets the number of Mov instructions.
ID3D12ShaderReflection::GetNumInterfaceSlots	Gets the number of interface slots in a shader.
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::GetResourceBindingDesc	Gets a description of how a resource is bound to a shader.
ID3D12ShaderReflection::GetResourceBindingDescByName	Gets a description of how a resource is bound to a shader.

METHOD	DESCRIPTION
ID3D12ShaderReflection::GetThreadGroupSize	Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid.
ID3D12ShaderReflection::GetVariableByName	Gets a variable by name.
ID3D12ShaderReflection::IsSampleFrequencyShader	Indicates whether a shader is a sample frequency shader.

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

Target Platform	Windows
Header	<code>d3d12shader.h</code>

See also

[IUnknown](#)

[Shader Interfaces](#)

ID3D12ShaderReflection::GetBitwiseInstructionCount method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of bitwise instructions.

Syntax

```
UINT GetBitwiseInstructionCount();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetConstantBufferByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets a constant buffer by index.

Syntax

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByIndex(  
    UINT Index  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetConstantBufferByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a constant buffer by name.

Syntax

```
ID3D12ShaderReflectionConstantBuffer * GetConstantBufferByName(  
    LPCSTR Name  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetConversionInstructionCount method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of conversion instructions.

Syntax

```
UINT GetConversionInstructionCount();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader description.

Syntax

```
HRESULT GetDesc(  
    D3D12_SHADER_DESC *pDesc  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetGSInputPrimitive method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the geometry-shader input-primitive description.

Syntax

```
D3D_PRIMITIVE GetGSInputPrimitive();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetInputParameterDesc method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets an input-parameter description for a shader.

Syntax

```
HRESULT GetInputParameterDesc(  
    UINT             ParameterIndex,  
    D3D12_SIGNATURE_PARAMETER_DESC *pDesc  
)
```

Parameters

ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

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

Target Platform	Windows
Header	d3d12shader.h

See also

ID3D12ShaderReflection

ID3D12ShaderReflection::GetMinFeatureLevel method

6/30/2020 • 2 minutes to read • [Edit Online](#)

Gets the minimum feature level.

Syntax

```
HRESULT GetMinFeatureLevel(  
    D3D_FEATURE_LEVEL *pLevel  
>;
```

Parameters

arg1

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetMovcInstructionCount method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of Movc instructions.

Syntax

```
UINT GetMovcInstructionCount();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetMovInstructionCount method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of Mov instructions.

Syntax

```
UINT GetMovInstructionCount();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetNumInterfaceSlots method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of interface slots in a shader.

Syntax

```
UINT GetNumInterfaceSlots();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetOutputParameterDesc method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets an output-parameter description for a shader.

Syntax

```
HRESULT GetOutputParameterDesc(  
    UINT             ParameterIndex,  
    D3D12_SIGNATURE_PARAMETER_DESC *pDesc  
) ;
```

Parameters

ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

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

Target Platform	Windows
Header	d3d12shader.h

See also

ID3D12ShaderReflection

ID3D12ShaderReflection::GetPatchConstantParameterDesc method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets a patch-constant parameter description for a shader.

Syntax

```
HRESULT GetPatchConstantParameterDesc(  
    UINT ParameterIndex,  
    D3D12_SIGNATURE_PARAMETER_DESC *pDesc  
>;
```

Parameters

ParameterIndex

Type: [UINT](#)

A zero-based parameter index.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetRequiresFlags method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a group of flags that indicates the requirements of a shader.

Syntax

```
UINT64 GetRequiresFlags();
```

Parameters

This method has no parameters.

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 there are no requirements.

SHADER REQUIREMENT #DEFINE FLAG	DESCRIPTION
D3D_SHADER_REQUIRES_DOUBLES	Shader requires that the graphics driver and hardware support 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 <code>msad4</code> 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 VPAndRTArrayIndexFromAnyShaderFeedingRasterizerSupportedWithoutGSEmulation of the D3D12_FEATURE_DATA_D3D12_OPTIONS structure.

Remarks

Here is how the D3D12Shader.h header defines the shader requirements flags:

#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_RASTERIZER	0x00002000

Requirements

Target Platform	Windows
Header	d3d12shader.h

See also

[Checking Hardware Feature Support](#)

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetResourceBindingDesc method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets a description of how a resource is bound to a shader.

Syntax

```
HRESULT GetResourceBindingDesc(  
    UINT             ResourceIndex,  
    D3D12_SHADER_INPUT_BIND_DESC *pDesc  
)
```

Parameters

ResourceIndex

Type: [UINT](#)

A zero-based resource index.

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

Target Platform	Windows
Header	d3d12shader.h

See also

ID3D12ShaderReflection

ID3D12ShaderReflection::GetResourceBindingDescByName method

4/22/2020 • 2 minutes to read • [Edit Online](#)

Gets a description of how a resource is bound to a shader.

Syntax

```
HRESULT GetResourceBindingDescByName(  
    LPCSTR             Name,  
    D3D12_SHADER_INPUT_BIND_DESC *pDesc  
>;
```

Parameters

Name

Type: [LPCSTR](#)

The constant-buffer name of the resource.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetThreadGroupSize method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the sizes, in units of threads, of the X, Y, and Z dimensions of the shader's thread-group grid.

Syntax

```
UINT GetThreadGroupSize(  
    UINT *pSizeX,  
    UINT *pSizeY,  
    UINT *pSizeZ  
) ;
```

Parameters

`pSizeX`

Type: [UINT*](#)

A pointer to the size, in threads, of the x-dimension of the thread-group grid. The maximum size is 1024.

`pSizeY`

Type: [UINT*](#)

A pointer to the size, in threads, of the y-dimension of the thread-group grid. The maximum size is 1024.

`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.

```
*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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::GetVariableByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a variable by name.

Syntax

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    LPCSTR Name  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflection::IsSampleFrequencyShader method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates whether a shader is a sample frequency shader.

Syntax

```
BOOL IsSampleFrequencyShader();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflection](#)

ID3D12ShaderReflectionConstantBuffer interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

This shader-reflection interface provides access to a constant buffer.

Methods

The **ID3D12ShaderReflectionConstantBuffer** interface has these methods.

METHOD	DESCRIPTION
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

Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

ID3D12ShaderReflectionConstantBuffer::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a constant-buffer description.

Syntax

```
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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

ID3D12ShaderReflectionConstantBuffer::GetVariableByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-reflection variable by index.

Syntax

```
ID3D12ShaderReflectionVariable * GetVariableByIndex(  
    UINT Index  
) ;
```

Parameters

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 D3DCCompiler_xx.dll.

Requirements

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

ID3D12ShaderReflectionConstantBuffer::GetVariableByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-reflection variable by name.

Syntax

```
ID3D12ShaderReflectionVariable * GetVariableByName(  
    LPCSTR Name  
) ;
```

Parameters

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 D3DCCompiler_xx.dll.

Requirements

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionConstantBuffer](#)

ID3D12ShaderReflectionType interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

This shader-reflection interface provides access to variable type.

Methods

The ID3D12ShaderReflectionType interface has these methods.

METHOD	DESCRIPTION
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.
ID3D12ShaderReflectionType::GetSubType	Gets the base class of a class.
ID3D12ShaderReflectionType::ImplementsInterface	Indicates whether a class type implements an interface.
ID3D12ShaderReflectionType::IsEqual	Indicates whether two ID3D12ShaderReflectionType Interface pointers have the same underlying type.
ID3D12ShaderReflectionType::IsOfType	Indicates whether a variable is of the specified type.

Remarks

To 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

Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

ID3D12ShaderReflectionType::GetBaseClass method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets an [ID3D12ShaderReflectionType Interface](#) interface containing the variable base class type.

Syntax

```
ID3D12ShaderReflectionType * GetBaseClass();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
HRESULT GetDesc(  
    D3D12_SHADER_TYPE_DESC *pDesc  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetInterfaceByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets an interface by index.

Syntax

```
ID3D12ShaderReflectionType * GetInterfaceByIndex(  
    UINT uIndex  
) ;
```

Parameters

`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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetMemberTypeByIndex method

5/8/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-reflection-variable type by index.

Syntax

```
ID3D12ShaderReflectionType * GetMemberTypeByIndex(  
    UINT Index  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetMemberTypeByName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-reflection-variable type by name.

Syntax

```
ID3D12ShaderReflectionType * GetMemberTypeByName(  
    LPCSTR Name  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetMemberTypeName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-reflection-variable type.

Syntax

```
LPCSTR GetMemberTypeName(  
    UINT Index  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetNumInterfaces method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the number of interfaces.

Syntax

```
UINT GetNumInterfaces();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::GetSubType method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets the base class of a class.

Syntax

```
ID3D12ShaderReflectionType * GetSubType();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::ImplementsInterface method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates whether a class type implements an interface.

Syntax

```
HRESULT ImplementsInterface(  
    ID3D12ShaderReflectionType *pBase  
) ;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::IsEqual method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates whether two [ID3D12ShaderReflectionType Interface](#) pointers have the same underlying type.

Syntax

```
HRESULT IsEqual(  
    ID3D12ShaderReflectionType *pType  
>;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionType::IsOfType method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Indicates whether a variable is of the specified type.

Syntax

```
HRESULT IsOfType(  
    ID3D12ShaderReflectionType *pType  
>;
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionType](#)

ID3D12ShaderReflectionVariable interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

This shader-reflection interface provides access to a variable.

Methods

The **ID3D12ShaderReflectionVariable** interface has these methods.

METHOD	DESCRIPTION
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::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

Target Platform	Windows
Header	d3d12shader.h

See also

[Shader Interfaces](#)

ID3D12ShaderReflectionVariable::GetBuffer method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Returns the [ID3D12ShaderReflectionConstantBuffer](#) of the present [ID3D12ShaderReflectionVariable](#).

Syntax

```
ID3D12ShaderReflectionConstantBuffer * GetBuffer();
```

Parameters

This method has no parameters.

Return value

Type: [ID3D12ShaderReflectionConstantBuffer*](#)

Returns a pointer to the [ID3D12ShaderReflectionConstantBuffer](#) of the present [ID3D12ShaderReflectionVariable](#).

Requirements

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

ID3D12ShaderReflectionVariable::GetDesc method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-variable description.

Syntax

```
HRESULT GetDesc(  
    D3D12_SHADER_VARIABLE_DESC *pDesc  
)
```

Parameters

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

ID3D12ShaderReflectionVariable::GetInterfaceSlot method

5/27/2020 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
UINT GetInterfaceSlot(  
    UINT uArrayIndex  
) ;
```

Parameters

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 an 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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

ID3D12ShaderReflectionVariable::GetType method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets a shader-variable type.

Syntax

```
ID3D12ShaderReflectionType * GetType();
```

Parameters

This method has no parameters.

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

Target Platform	Windows
Header	d3d12shader.h

See also

[ID3D12ShaderReflectionVariable](#)

directml.h header

5/13/2020 • 17 minutes to read • [Edit Online](#)

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

- [Direct3D 12 Graphics](#) directml.h contains the following programming interfaces:

Interfaces

TITLE	DESCRIPTION
IDMLBindingTable	Wraps a range of an application-managed descriptor heap, and is used by DirectML to create bindings for resources. To create this object, call IDMLDevice::CreateBindingTable.
IDMLCommandRecorder	Records dispatches of DirectML work into a Direct3D 12 command list.
IDMLCompiledOperator	Represents a compiled, efficient form of an operator suitable for execution on the GPU. To create this object, call IDMLDevice::CompileOperator.
IDMLDebugDevice	Controls the DirectML debug layers.
IDMLDevice	Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects.
IDMLDeviceChild	An interface implemented by all objects created from the DirectML device.
IDMLDispatchable	Implemented by objects that can be recorded into a command list for dispatch on the GPU, using IDMLCommandRecorder::RecordDispatch.
IDMLObject	An interface from which IDMLDevice and IDMLDeviceChild inherit directly (and all other interfaces, indirectly).
IDMLOperator	Represents a DirectML operator.
IDMLOperatorInitializer	Represents a specialized object whose purpose is to initialize compiled operators. To create an instance of this object, call IDMLDevice::CreateOperatorInitializer.
IDMLPageable	Implemented by objects that can be evicted from GPU memory, and hence that can be supplied to IDMLDevice::Evict and IDMLDevice::MakeResident.

Functions

TITLE	DESCRIPTION
DMLCreateDevice	Creates a DirectML device for a given Direct3D 12 device.

Structures

TITLE	DESCRIPTION
DML_ACTIVATION_ELU_OPERATOR_DESC	Describes a DirectML operator that performs an exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } (\exp(x) - 1) * \alpha$.
DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC	Describes a DirectML activation operator that performs a hard sigmoid function on every element in the input, $f(x) = \max(0, \min(\alpha * x + \beta, 1))$.
DML_ACTIVATION_HARDMAX_OPERATOR_DESC	Describes a DirectML activation operator that performs a hardmax function on the input, $f(x) = \text{if } x_{\text{j}} == \max(x) \text{ then } 1 \text{ else } 0$ (but only for the first element in the axis).
DML_ACTIVATION_IDENTITY_OPERATOR_DESC	Describes a DirectML activation operator that performs the identity function $f(x) = x$. The operator effectively copies its input tensor to the output.
DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a leaky rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } x * \alpha$.
DML_ACTIVATION_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs a linear activation function on every element in the input, $f(x) = \alpha * x + \beta$.
DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC	Describes a DirectML operator that performs a log-of-softmax activation function on the input, $f(x_i) = \log(\exp(x_i - \max(X)) / \sum(\exp(X - \max(X)))) = (x_i - \max(X)) - \log(\sum(\exp(x - \max(X))))$.
DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a parameterized rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } \text{slope} * x$.
DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC	Describes a DirectML operator that performs a parametric softplus activation function on every element in the input, $f(x) = \alpha * \text{log}(1 + \exp(\beta * x))$.
DML_ACTIVATION_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \max(0, x)$.
DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC	Describes a DirectML operator that performs a scaled exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x > 0 \text{ then } \gamma * x \text{ else } \gamma * (\alpha * e^x - \alpha)$.

TITLE	DESCRIPTION
DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC	Describes a DirectML operator that performs a scaled hyperbolic tangent activation function on every element in the input, $f(x) = \alpha * \tanh(\beta * x)$.
DML_ACTIVATION_SHRINK_OPERATOR_DESC	Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ACTIVATION_SIGMOID_OPERATOR_DESC	Describes a DirectML operator that performs a sigmoid activation function on every element in the input, $f(x) = 1 / (1 + \exp(-x))$.
DML_ACTIVATION_SOFTMAX_OPERATOR_DESC	Describes a DirectML operator that performs a softmax activation function on the input, $f(x_i) = \exp(x_i) / \sum(\exp(X)) = \exp(x_i - \max(X)) / \sum(\exp(x - \max(X)))$.
DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC	Describes a DirectML operator that performs a softplus activation function on every element in the input, $f(x) = \ln(1 + \exp(x))$.
DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC	Describes a DirectML operator that performs a softsign activation function on every element in the input, $f(x) = x / (1 + \text{abs}(x))$.
DML_ACTIVATION_TANH_OPERATOR_DESC	Describes a DirectML operator that performs a hyperbolic tangent activation function on every element in the input, $f(x) = (1 - \exp(-2 * x)) / (1 + \exp(-2 * x))$.
DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC	Describes a DirectML operator that performs a thresholded rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x > \alpha \text{ then } x \text{ else } 0$.
DML_AVERAGE_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs an average pooling function on the input, $y = (x_1 + x_2 + \dots) / \text{pool_size}$.
DML_BATCH_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a batch normalization function on the input, $y = \text{scale} * (x - \text{batchMean}) / \sqrt{\text{batchVariance} + \text{epsilon}} + \text{bias}$.
DML_BINDING_DESC	Contains the description of a binding so that you can add it to the binding table via a call to one of the IDMLBindingTable methods.
DML_BINDING_PROPERTIES	Contains information about the binding requirements of a particular compiled operator, or operator initializer. This struct is retrieved from IDMLDispatchable::GetBindingProperties.
DML_BINDING_TABLE_DESC	Specifies parameters to IDMLDevice::CreateBindingTable and IDMLBindingTable::Reset.
DML_BUFFER_ARRAY_BINDING	Specifies a resource binding that is an array of individual buffer bindings.
DML_BUFFER_BINDING	Specifies a resource binding described by a range of bytes in a Direct3D 12 buffer, represented by an offset and size into an ID3D12Resource.

TITLE	DESCRIPTION
DML_BUFFER_TENSOR_DESC	Describes a tensor that will be stored in a Direct3D 12 buffer resource.
DML_CAST_OPERATOR_DESC	Describes a DirectML data reorganization operator that performs the cast function $f(x) = \text{cast}(x)$, casting each element in the input to the data type of the output tensor, and storing the result in the corresponding element in the output.
DML_CONVOLUTION_OPERATOR_DESC	Describes a DirectML matrix multiplication operator that performs a convolution function on the input, $\text{out}[j] = x[i]*w[0] + x[i+1]*w[1] + x[i+2]*w[2] + \dots + x[i+k]*w[k] + \text{bias}$.
DML_DEPTH_TO_SPACE_OPERATOR_DESC	Describes a DirectML data reorganization operator that rearranges (permutes) data from depth into blocks of spatial data.
DML_DIAGONAL_MATRIX_OPERATOR_DESC	Describes a DirectML math operator that generates an identity-like matrix with ones on the major diagonal and zeros everywhere else.
DML_ELEMENT_WISE_ABS_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise absolute value function $f(x) = \text{abs}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The absolute value of x is its magnitude without regard to its sign.
DML_ELEMENT_WISE_ACOS_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arccosine function $f(x) = \text{acos}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic cosine function $f(x) = \log(x + \sqrt{x * x - 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ADD_OPERATOR_DESC	Describes a DirectML math operator that performs the function of adding every element in $ATensor$ to its corresponding element in $BTensor$, $f(a, b) = a + b$.
DML_ELEMENT_WISE_ADD1_OPERATOR_DESC	Describes a DirectML math operator that performs the function of adding every element in $ATensor$ to its corresponding element in $BTensor$, with the option for fused activation.
DML_ELEMENT_WISE_ASIN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arcsine function $f(x) = \text{asin}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ASINH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic sine function $f(x) = \log(x + \sqrt{x * x + 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_ATAN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise arctangent function $f(x) = \text{atan}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

TITLE	DESCRIPTION
DML_ELEMENT_WISE_ATANH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = (\log((1 + x) / (1 - x)) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_CEIL_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise ceiling function $f(x) = \text{ceil}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The ceiling of x is the smallest integer that is greater than or equal to x .
DML_ELEMENT_WISE_CLIP_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise clip function $f(x) = \text{clamp}(x * \text{scale} + \text{bias}, \text{minValue}, \text{maxValue})$, where the scale and bias terms are optional, and where $\text{clamp}(x) = \min(\text{maxValue}, \max(\text{minValue}, x))$.
DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC	Describes a DirectML operator that performs the element-wise constant power function $f(x) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional. The power function raises every element in the input to the power of the exponent.
DML_ELEMENT_WISE_COS_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise cosine function $f(x) = \cos(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_COSH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise hyperbolic cosine function $f(x) = ((e^x + e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs the linear dequantize function on every element in InputTensor with respect to its corresponding element in ScaleTensor and ZeroPointTensor, $f(\text{input}, \text{scale}, \text{zero_point}) = (\text{input} - \text{zero_point}) * \text{scale}$.
DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC	Describes a DirectML math operator that performs the function of dividing every element in ATensor by its corresponding element in BTensor, $f(a, b) = a / b$.
DML_ELEMENT_WISE_ERF_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_EXP_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise floor function $f(x) = \text{floor}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The floor of x is the largest integer that is less than or equal to x .

TITLE	DESCRIPTION
DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC	Describes a DirectML generic operator that performs the element-wise identity function $f(x) = x * \text{scale} + \text{bias}$. The operator effectively copies its input tensor to the output, while applying optional scale and bias terms.
DML_ELEMENT_WISE_IF_OPERATOR_DESC	Describes a DirectML math operator that essentially performs a ternary <code>if</code> statement.
DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC	Describes a DirectML math operator that determines, elementwise, whether the input is NaN.
DML_ELEMENT_WISE_LOG_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise natural logarithm function $f(x) = \log(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC	Describes a DirectML math operator that performs a logical AND function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a \&\& b$.
DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC	Describes a DirectML math operator that performs a logical equality function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a == b)$.
DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC	Describes a DirectML math operator that performs a logical greater-than function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a > b)$.
DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC	Describes a DirectML math operator that performs a logical less-than function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a < b)$.
DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC	Describes a DirectML math operator that performs a logical NOT function on every element in the input, $f(x) = !x$.
DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC	Describes a DirectML math operator that performs a logical OR function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a b$.
DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC	Describes a DirectML math operator that performs a logical exclusive OR (XOR) function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = a \text{ xor } b$.
DML_ELEMENT_WISE_MAX_OPERATOR_DESC	Describes a DirectML math reduction operator that performs a maximum function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = \max(a, b)$.
DML_ELEMENT_WISE_MEAN_OPERATOR_DESC	Describes a DirectML math reduction operator that performs an arithmetic mean function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = (a + b) / 2$.
DML_ELEMENT_WISE_MIN_OPERATOR_DESC	Describes a DirectML math reduction operator that performs a minimum function between every element in ATensor and its corresponding element in BTensor, $f(a, b) = \min(a, b)$.

TITLE	DESCRIPTION
DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC	Describes a DirectML math operator that performs the function of multiplying every element in ATensor by its corresponding element in BTensor, $f(a, b) = a * b$.
DML_ELEMENT_WISE_POW_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise power function $f(x, exponent) = \text{pow}(x * \text{scale} + \text{bias}, exponent)$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC	Describes a DirectML operator that performs the linear quantize function on every element in InputTensor with respect to its corresponding element in ScaleTensor and ZeroPointTensor, $f(\text{input}, \text{scale}, \text{zero_point}) = \text{clamp}(\text{round}(\text{input} / \text{scale}) + \text{zero_point}, 0, 255)$.
DML_ELEMENT_WISE_RECIP_OPERATOR_DESC	Describes a DirectML math operator that performs a reciprocal function on every element in the input, $f(x) = 1 / (x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SIGN_OPERATOR_DESC	Describes a DirectML operator that performs an elementwise shrink activation function on the input.
DML_ELEMENT_WISE_SIN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise sine function $f(x) = \sin(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SINH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise hyperbolic sine function $f(x) = ((e^x - e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SQRT_OPERATOR_DESC	Describes a DirectML math operator that performs a square root function on every element in the input, $f(x) = \sqrt{x * \text{scale} + \text{bias}}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC	Describes a DirectML math operator that performs the function of subtracting every element in BTensor from its corresponding element in ATensor, $f(a, b) = a - b$.
DML_ELEMENT_WISE_TAN_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise tangent function $f(x) = \tan(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_TANH_OPERATOR_DESC	Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = \tanh(x) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.
DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC	Describes a DirectML math operator that performs the element-wise threshold function $f(x) = \max(x * \text{scale} + \text{bias}, \text{min})$, where the scale and bias terms are optional. The threshold of x with respect to min is the larger of the two values.
DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT	Provides detail about whether a DirectML device supports a particular data type within tensors.

TITLE	DESCRIPTION
DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT	Used to query a DirectML device for its support for a particular data type within tensors.
DML_GATHER_OPERATOR_DESC	Describes a DirectML data reorganization operator which, when given a data tensor of rank $r \geq 1$, and an indices tensor of rank q , gathers the entries in the axis dimension of the data (by default, the outermost one is $\text{axis} == 0$) indexed by indices, and concatenates them in an output tensor of rank $q + (r - 1)$.
DML_GEMM_OPERATOR_DESC	Describes a DirectML operator that performs a general matrix multiplication function on the input, $y = \alpha * \text{transposeA}(A) * \text{transposeB}(B) + \beta * C$.
DML_GRU_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a (standard layers) one-layer gated recurrent unit (GRU) function on the input.
DML_JOIN_OPERATOR_DESC	Describes a DirectML operator that performs a join function on an array of input tensors.
DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a local response normalization (LRN) function on the input, $y = x / (\text{bias} + (\alpha / \text{size}) * \sum(x_i^2 \text{ for every } x_i \text{ in the local region}))^\beta$.
DML_LP_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs an Lp-normalization function along the specified axis of the input tensor.
DML_LP_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs an Lp pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = (x_1^p + x_2^p + \dots + x_n^p)^{1/p}$ where $X \rightarrow Y$ reduced for each kernel.
DML_LSTM_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a one-layer long short term memory (LSTM) function on the input.
DML_MAX_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = \max(x_1 + x_2 + \dots + x_{\text{pool_size}})$.
DML_MAX_POOLING1_OPERATOR_DESC	Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths).
DML_MAX_UNPOOLING_OPERATOR_DESC	Describes a DirectML operator that fills the output tensor of the given shape (either explicit, or the input shape plus padding) with zeros, then writes each value from the input tensor into the output tensor at the element offset from the corresponding indices array.
DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC	Describes a DirectML operator that performs a mean variance normalization function on the input tensor.

TITLE	DESCRIPTION
DML_ONE_HOT_OPERATOR_DESC	Describes a DirectML operator that generates a tensor with each element filled with two values—either an 'on' or an 'off' value.
DML_OPERATOR_DESC	A generic container for an operator description. You construct DirectML operators using the parameters specified in this struct. See IDMLDevice::CreateOperator for additional details.
DML_PADDING_OPERATOR_DESC	Describes a DirectML data reorganization operator that inflates the input tensor with zeroes (or some other value) on the edges.
DML_REDUCE_OPERATOR_DESC	Describes a DirectML operator that performs the specified reduction function on the input.
DML_RESAMPLE_OPERATOR_DESC	Describes a DirectML operator that resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size.
DML_RNN_OPERATOR_DESC	Describes a DirectML deep learning operator that performs a one-layer simple recurrent neural network (RNN) function on the input.
DML_ROI_POOLING_OPERATOR_DESC	Describes a DirectML operator that performs a pooling function across the input tensor (according to regions of interest, or ROIs).
DML_SCALE_BIAS	Contains the values of scale and bias terms supplied to a DirectML operator. Scale and bias have the effect of applying the function $g(x) = x * \text{Scale} + \text{Bias}$.
DML_SCATTER_OPERATOR_DESC	Describes a DirectML operator that copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor.
DML_SIZE_2D	Contains values that can represent the size (as supplied to a DirectML operator) of a 2-D plane of elements within a tensor, or a 2-D scale, or any 2-D width/height value.
DML_SLICE_OPERATOR_DESC	Describes a DirectML data reorganization operator that produces a slice of the input tensor along multiple axes.
DML_SPACE_TO_DEPTH_OPERATOR_DESC	Describes a DirectML data reorganization operator that rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.
DML_SPLIT_OPERATOR_DESC	Describes a DirectML data reorganization operator that splits the input tensor into multiple output tensors, along the specified axis.
DML_TENSOR_DESC	A generic container for a DirectML tensor description.

TITLE	DESCRIPTION
DML_TILE_OPERATOR_DESC	Describes a DirectML data reorganization operator that constructs an output tensor by tiling the input tensor.
DML_TOP_K_OPERATOR_DESC	Describes a DirectML reduction operator that retrieves the top K elements along a specified axis.
DML_UPSAMPLE_2D_OPERATOR_DESC	Describes a DirectML imaging operator that upsamples the image contained in the input tensor. Each dimension value of the output tensor is <code>output_dimension = floor(input_dimension * scale)</code> .
DML_VALUE_SCALE_2D_OPERATOR_DESC	Describes a DirectML operator that performs an element-wise scale-and-bias function on the values in the input tensor.

Enumerations

TITLE	DESCRIPTION
DML_BINDING_TYPE	Defines constants that specify the nature of the resource(s) referred to by a binding description (a <code>DML_BINDING_DESC</code> structure).
DML_CONVOLUTION_DIRECTION	Defines constants that specify a direction for the DirectML convolution operator (as described by the <code>DML_CONVOLUTION_OPERATOR_DESC</code> structure).
DML_CONVOLUTION_MODE	Defines constants that specify a mode for the DirectML convolution operator (as described by the <code>DML_CONVOLUTION_OPERATOR_DESC</code> structure).
DML_CREATE_DEVICE_FLAGS	Supplies additional device creation options to <code>DMLCreateDevice</code> . Values can be bitwise OR'd together.
DML_EXECUTION_FLAGS	Supplies options to DirectML to control execution of operators. These flags can be bitwise OR'd together to specify multiple flags at once.
DML_FEATURE	Defines a set of optional features and capabilities that can be queried from the DirectML device.
DML_INTERPOLATION_MODE	Defines constants that specify a mode for the DirectML upsample 2-D operator (as described by the <code>DML_UPSAMPLE_2D_OPERATOR_DESC</code> structure).
DML_MATRIX_TRANSFORM	Defines constants that specify a matrix transform to be applied to a DirectML tensor.
DML_OPERATOR_TYPE	Defines the type of an operator description.
DML_PADDING_MODE	Defines constants that specify a mode for the DirectML pad operator (as described by the <code>DML_PADDING_OPERATOR_DESC</code> structure).

TITLE	DESCRIPTION
DML_RECURRENT_NETWORK_DIRECTION	Defines constants that specify a direction for a recurrent DirectML operator.
DML_REDUCE_FUNCTION	Defines constants that specify the specific reduction algorithm to use for the DirectML reduce operator (as described by the DML_REDUCE_OPERATOR_DESC structure).
DML_TENSOR_DATA_TYPE	Specifies the data type of the values in a tensor. DirectML operators may not support all data types; see the documentation for each specific operator to find which data types it supports.
DML_TENSOR_FLAGS	Specifies additional options in a tensor description. Values can be bitwise OR'd together.
DML_TENSOR_TYPE	Identifies a type of tensor description.

DML_ACTIVATION_ELU_OPERATOR_DESC structure

4/28/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } (\exp(x) - 1) * \text{alpha}$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_ELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The coefficient of ELU.

Requirements

Header	directml.h
---------------	------------

DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML activation operator that performs a hard sigmoid function on every element in the input, $f(x) = \max(0, \min(\alpha * x + \beta, 1))$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
    FLOAT Beta;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha.

Beta

Type: **FLOAT**

The value of beta.

Requirements

Header	directml.h
---------------	------------

DML_ACTIVATION_HARDMAX_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML activation operator that performs a hardmax function on the input, $f(x) = \text{if } x_i == \max(x) \text{ then } 1 \text{ else } 0$ (but only for the first element in the axis).

Syntax

```
struct DML_ACTIVATION_HARDMAX_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Remarks

The operator computes the hardmax (1 for the first maximum value, and 0 for all others) values for each layer in the batch of the given input. The input is a 2-D tensor (Tensor) of size (batch_size x input_feature_dimensions). The output tensor has the same shape and contains the hardmax values of the corresponding input.

Requirements

Header

directml.h

DML_ACTIVATION_IDENTITY_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML activation operator that performs the identity function $f(x) = x$. The operator effectively copies its input tensor to the output.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_IDENTITY_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a leaky rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else } x * \alpha$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The coefficient of ReLU.

Requirements

Header	directml.h
--------	------------

See also

[DML_ACTIVATION_RELU_OPERATOR_DESC](#)

DML_ACTIVATION_LINEAR_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a linear activation function on every element in the input, $f(x) = \text{alpha} * x + \text{beta}$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
    FLOAT Beta;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha.

Beta

Type: **FLOAT**

The value of beta.

Requirements

Header	directml.h
---------------	------------

DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a log-of-softmax activation function on the input, $f(x_i) = \log(\exp(x_i - \max(X)) / \sum(\exp(x_i - \max(X)))) = (x_i - \max(X)) - \log(\sum(\exp(x_i - \max(X))))$.

Syntax

```
struct DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

See also

[DML_ACTIVATION_SOFTMAX_OPERATOR_DESC](#)

DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a parameterized rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x \geq 0 \text{ then } x \text{ else slope} * x$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *SlopeTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

SlopeTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the slope.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

See also

[DML_ACTIVATION_RELU_OPERATOR_DESC](#)

DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a parametric softplus activation function on every element in the input, $f(x) = \alpha * \text{log}(1 + \exp(\beta * x))$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Alpha;
    FLOAT Beta;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha.

Beta

Type: **FLOAT**

The value of beta.

Requirements

Header	directml.h
--------	------------

See also

[DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC](#)

DML_ACTIVATION_RELU_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \max(0, x)$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_RELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from. This operator supports in-place execution. That is, the supplied output tensor may be the same as the supplied input tensor.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a scaled exponential linear unit (ELU) activation function on every element in the input, $f(x) = \text{if } x > 0 \text{ then } \gamma * x \text{ else } \gamma * (\alpha * e^x - \alpha)$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
    FLOAT Gamma;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha. You can use a default value of 1.6732.

Gamma

Type: **FLOAT**

The value of gamma. You can use a default value of 1.0507.

Requirements

Header

directml.h

See also

DML_ACTIVATION_ELU_OPERATOR_DESC

DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a scaled hyperbolic tangent activation function on every element in the input, $f(x) = \text{alpha} * \tanh(\text{beta} * x)$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Alpha;  
    FLOAT Beta;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Alpha

Type: **FLOAT**

The value of alpha.

Beta

Type: **FLOAT**

The value of alpha.

Requirements

Header

directml.h

See also

DML_ACTIVATION_TANH_OPERATOR_DESC

DML_ACTIVATION_SHRINK_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an elementwise shrink activation function on the input.

```
For each x in InputTensor
    if (x < -threshold)
        x = x + bias
    else if (x > threshold)
        x = x - bias
    else x = 0
```

Shrink has no relation to slice, nor resize, despite the naming similarity. It actually 'shrinks' per-element values. And it doesn't always shrink; but can instead grow, depending on the bias sign.

Syntax

```
struct DML_ACTIVATION_SHRINK_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT Bias;
    FLOAT Threshold;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Bias

Type: **FLOAT**

The value of bias. You can use a default value of 1.0.

Threshold

Type: **FLOAT**

The value of threshold.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ACTIVATION_SIGMOID_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a sigmoid activation function on every element in the input, $f(x) = 1 / (1 + \exp(-x))$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_SIGMOID_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ACTIVATION_SOFTMAX_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a softmax activation function on the input, $f(x_i) = \text{exp}(x_i) / \sum(\text{exp}(X)) = \exp(x_i - \max(X)) / \sum(\exp(x - \max(X)))$.

Syntax

```
struct DML_ACTIVATION_SOFTMAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header		directml.h

DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a softplus activation function on every element in the input, $f(x) = \ln(1 + e^{x})$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT             Steepness;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Steepness

Type: **FLOAT**

The steepness value.

Requirements

Header	directml.h
--------	------------

DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a softsign activation function on every element in the input, $f(x) = x / (1 + \text{abs}(x))$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ACTIVATION_TANH_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a hyperbolic tangent activation function on every element in the input, $f(x) = (1 - \exp(-2 * x)) / (1 + \exp(-2 * x))$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_TANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a thresholded rectified linear unit (ReLU) activation function on every element in the input, $f(x) = \text{if } x > \text{alpha} \text{ then } x \text{ else } 0$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    FLOAT                 Alpha;
};
```

Members

`InputTensor`

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

`OutputTensor`

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

`Alpha`

Type: **FLOAT**

The coefficient of ReLU.

Requirements

Header	directml.h
--------	------------

See also

[DML_ACTIVATION_SOFTMAX_OPERATOR_DESC](#)

DML_AVERAGE_POOLING_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an average pooling function on the input, $y = (x_1 + x_2 + \dots) / \text{pool_size}$.

Syntax

```
struct DML_AVERAGE_POOLING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT DimensionCount;
    const UINT *Strides;
    const UINT *WindowSize;
    const UINT *StartPadding;
    const UINT *EndPadding;
    BOOL IncludePadding;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

DimensionCount

Type: **UINT**

The number of dimensions. This field determines the size of the *Strides*, *WindowSize*, *StartPadding*, and *EndPadding* arrays.

Strides

Type: **const UINT***

A pointer to a constant array of **UINT** containing the lengths of the strides of the tensor.

WindowSize

Type: **const UINT***

A pointer to a constant array of **UINT** containing the lengths of the pooling windows.

StartPadding

Type: **const UINT***

A pointer to a constant array of **UINT** containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

EndPadding

Type: **const UINT***

A pointer to a constant array of **UINT** containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

IncludePadding

Type: **BOOL**

TRUE if pad pixels should be included when calculating values for the edges, otherwise FALSE.

Requirements

Header	directml.h

DML_BATCH_NORMALIZATION_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a batch normalization function on the input, $y = \text{scale} * (x - \text{batchMean}) / \sqrt{\text{batchVariance} + \text{epsilon}} + \text{bias}$.

Syntax

```
struct DML_BATCH_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *MeanTensor;
    const DML_TENSOR_DESC *VarianceTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *OutputTensor;
    BOOL Spatial;
    FLOAT Epsilon;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

MeanTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing batchMean.

VarianceTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the batchVariance.

ScaleTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the scale.

BiasTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the bias.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

`Spatial`

Type: [BOOL](#)

TRUE to specify that locations are spatial, otherwise FALSE.

`Epsilon`

Type: [FLOAT](#)

The epsilon value to use to avoid division by zero.

`FusedActivation`

Type: `const DML_OPERATOR_DESC*`

An optional pointer to a constant [DML_OPERATOR_DESC](#) containing the fused activation layer.

Requirements

Header	
<code>directml.h</code>	

DML_BINDING_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Contains the description of a binding so that you can add it to the binding table via a call to one of the [IDMLBindingTable](#) methods.

A binding can refer to an input or an output tensor resource, or to a persistent or a temporary resource, and there are methods on [IDMLBindingTable](#) to bind each kind. The type of the structure pointed to by *Desc* depends on the value of *Type*.

Syntax

```
struct DML_BINDING_DESC {
    DML_BINDING_TYPE Type;
    const void        *Desc;
};
```

Members

Type

Type: [DML_BINDING_TYPE](#)

A [DML_BINDING_TYPE](#) specifying the type of the binding; whether it refers to a single buffer, or to an array of buffers.

Desc

Type: [const void*](#)

A pointer to a constant structure whose type depends on the value *Type*. If *Type* is [DML_BINDING_TYPE_BUFFER](#), then *Desc* should point to a [DML_BUFFER_BINDING](#). If *Type* is [DML_BINDING_TYPE_BUFFER_ARRAY](#), then *Desc* should point to a [DML_BUFFER_ARRAY_BINDING](#).

Requirements

Header	directml.h

See also

[Binding in DirectML](#)

[IDMLBindingTable::BindInputs](#)

[IDMLBindingTable::BindOutputs](#)

[IDMLBindingTable::BindPersistentResource](#)

[IDMLBindingTable::BindTemporaryResource](#)

DML_BINDING_PROPERTIES structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Contains information about the binding requirements of a particular compiled operator, or operator initializer. This struct is retrieved from [IDMLDispatchable::GetBindingProperties](#).

Syntax

```
struct DML_BINDING_PROPERTIES {
    UINT RequiredDescriptorCount;
    UINT64 TemporaryResourceSize;
    UINT64 PersistentResourceSize;
};
```

Members

`RequiredDescriptorCount`

Type: [UINT](#)

The minimum size, in descriptors, of the binding table required for a particular dispatchable object (an operator initializer, or a compiled operator).

`TemporaryResourceSize`

Type: [UINT64](#)

The minimum size in bytes of the temporary resource that must be bound to the binding table for a particular dispatchable object. A value of zero means that a temporary resource is not required.

`PersistentResourceSize`

Type: [UINT64](#)

The minimum size in bytes of the persistent resource that must be bound to the binding table for a particular dispatchable object. Persistent resources must be supplied during initialization of a compiled operator (where it is bound as an output of the operator initializer) as well as during execution. A value of zero means that a persistent resource is not required. Only compiled operators have persistent resources—operator initializers always return a value of 0 for this member.

Requirements

Header	directml.h

See also

[Binding in DirectML](#)

DML_BINDING_TABLE_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies parameters to [IDMLDevice::CreateBindingTable](#) and [IDMLBindingTable::Reset](#).

Syntax

```
struct DML_BINDING_TABLE_DESC {
    IDMLDispatchable           *Dispatchable;
    D3D12_CPU_DESCRIPTOR_HANDLE CPUDescriptorHandle;
    D3D12_GPU_DESCRIPTOR_HANDLE GPUDescriptorHandle;
    UINT                        SizeInDescriptors;
};
```

Members

Dispatchable

Type: [IDMLDispatchable*](#)

A pointer to an [IDMLDispatchable](#) interface representing the dispatchable object (an operator initializer, or a compiled operator) for which this binding table will represent the bindings—either an [IDMLCompiledOperator](#) or an [IDMLOperatorInitializer](#). The binding table maintains a strong reference to this interface pointer. This value may not be null.

CPUDescriptorHandle

Type: [D3D12_CPU_DESCRIPTOR_HANDLE](#)

A valid CPU descriptor handle representing the start of a range into a constant buffer view (CBV)/shader resource view (SRV)/ unordered access view (UAV) descriptor heap into which DirectML may write descriptors.

GPUDescriptorHandle

Type: [D3D12_GPU_DESCRIPTOR_HANDLE](#)

A valid GPU descriptor handle representing the start of a range into a constant buffer view (CBV)/shader resource view (SRV)/ unordered access view (UAV) descriptor heap that DirectML may use to bind resources to the pipeline.

SizeInDescriptors

Type: [UINT](#)

The size of the binding table, in descriptors. This is the maximum number of descriptors that DirectML is permitted to write, from the start of both the supplied CPU and GPU descriptor handles. Call [IDMLDispatchable::GetBindingProperties](#) to determine the number of descriptors required to execute a dispatchable object.

Requirements

Header	directml.h

See also

[Binding in DirectML](#)

DML_BINDING_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify the nature of the resource(s) referred to by a binding description (a [DML_BINDING_DESC](#) structure).

Syntax

```
typedef enum DML_BINDING_TYPE {  
    DML_BINDING_TYPE_NONE,  
    DML_BINDING_TYPE_BUFFER,  
    DML_BINDING_TYPE_BUFFER_ARRAY  
} ;
```

Constants

DML_BINDING_TYPE_NONE	Indicates that no resources are to be bound.
DML_BINDING_TYPE_BUFFER	Specifies a binding that binds a single buffer to the binding table. The corresponding binding desc type is DML_BUFFER_BINDING .
DML_BINDING_TYPE_BUFFER_ARRAY	Specifies a binding that binds an array of buffers to the binding table. The corresponding binding desc type is DML_BUFFER_ARRAY_BINDING .

Requirements

Header	directml.h
--------	------------

See also

[Binding in DirectML](#), [DML_BINDING_DESC](#)

DML_BUFFER_ARRAY_BINDING structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a resource binding that is an array of individual buffer bindings.

Syntax

```
struct DML_BUFFER_ARRAY_BINDING {
    UINT             BindingCount;
    const DML_BUFFER_BINDING *Bindings;
};
```

Members

BindingCount

Type: [UINT](#)

The number of individual buffer ranges to bind to this slot. This field determines the size of the *Bindings* array.

Bindings

Type: [const DML_BUFFER_BINDING*](#)

The individual buffer ranges to bind.

Requirements

Header

directml.h

See also

[Binding in DirectML](#)

DML_BUFFER_BINDING structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies a resource binding described by a range of bytes in a Direct3D 12 buffer, represented by an offset and size into an [ID3D12Resource](#).

Syntax

```
struct DML_BUFFER_BINDING {  
    ID3D12Resource *Buffer;  
    UINT64          Offset;  
    UINT64          SizeInBytes;  
};
```

Members

Buffer

Type: [ID3D12Resource*](#)

An optional pointer to an [ID3D12Resource](#) interface representing a buffer. The resource must have dimension [D3D12_RESOURCE_DIMENSION_BUFFER](#), and the range described by this struct must lie within the bounds of the buffer. You may supply `nullptr` for this member to indicate 'no binding'.

Offset

Type: [UINT64](#)

The offset, in bytes, from the start of the buffer where the range begins. This offset must be aligned to a multiple of [DML_MINIMUM_BUFFER_TENSOR_ALIGNMENT](#) or the [GuaranteedBaseOffsetAlignment](#) supplied as part of the [DML_BUFFER_TENSOR_DESC](#).

SizeInBytes

Type: [UINT64](#)

The size of the range, in bytes.

Requirements

Header	directml.h

See also

[Binding in DirectML](#)

DML_BUFFER_TENSOR_DESC structure

5/27/2020 • 3 minutes to read • [Edit Online](#)

Describes a tensor that will be stored in a Direct3D 12 buffer resource. The corresponding tensor type is [DML_TENSOR_TYPE_BUFFER](#), and the corresponding binding type is [DML_BINDING_TYPE_BUFFER](#).

Syntax

```
struct DML_BUFFER_TENSOR_DESC {
    DML_TENSOR_DATA_TYPE DataType;
    DML_TENSOR_FLAGS      Flags;
    UINT                  DimensionCount;
    const UINT             *Sizes;
    const UINT             *Strides;
    UINT64                TotalTensorSizeInBytes;
    UINT                  GuaranteedBaseOffsetAlignment;
};
```

Members

DataType

Type: [DML_TENSOR_DATA_TYPE](#)

The type of the values in the tensor.

Flags

Type: [DML_TENSOR_FLAGS](#)

Specifies additional options for the tensor.

DimensionCount

Type: [UINT](#)

The number of dimensions of the tensor. This member determines the size of the *Sizes* and *Strides* arrays (if provided). In DirectML, all buffer tensors must have a *DimensionCount* of either 4 or 5. Not all operators support a *DimensionCount* of 5.

Sizes

Type: [const UINT*](#)

The size, in elements, of each dimension in the tensor. Specifying a size of zero in any dimension is invalid, and will result in an error. The *Sizes* member is always specified in the order {N, C, H, W} if *DimensionCount* is 4, and {N, C, D, H, W} if *DimensionCount* is 5.

Strides

Type: [const UINT*](#)

Optional. Determines the number of elements (not bytes) to linearly traverse in order to reach the next element in that dimension. For example, a stride of 5 in dimension 1 means that the distance between elements (n) and (n+1) in that dimension is 5 elements when traversing the buffer linearly. The *Strides* member is always specified in the

order {N, C, H, W} if *DimensionCount* is 4, and {N, C, D, H, W} if *DimensionCount* is 5.

Strides can be used to express broadcasting (by specifying a stride of 0) as well as padding (for example, by using a stride larger than the physical size of a row, to pad the end of a row).

If *Strides* is not specified, each dimension in the tensor is considered to be contiguously packed, with no additional padding.

`TotalTensorSizeInBytes`

Type: [UINT64](#)

Defines a minimum size in bytes for the buffer that will contain this tensor. *TotalTensorSizeInBytes* must be at least as large as the minimum implied size given the sizes, strides, and data type of the tensor. You can calculate the minimum implied size by calling the [DMLCalcBufferTensorSize](#) utility free function.

Providing a *TotalTensorSizeInBytes* that is larger than the minimum implied size may enable additional optimizations by allowing DirectML to elide bounds checking in some cases if the *TotalTensorSizeInBytes* defines sufficient padding beyond the end of the tensor data.

When binding this tensor, the size of the buffer range must be at least as large as the *TotalTensorSizeInBytes*. For output tensors, this has the additional effect of permitting DirectML to write to any memory within the *TotalTensorSizeInBytes*. That is, your application mustn't assume that DirectML will preserve any padding bytes inside output tensors that are inside the *TotalTensorSizeInBytes*.

The total size of a buffer tensor may not exceed $(2^{32} - 1)$ elements—for example, 16GB for a [FLOAT32](#) tensor.

`GuaranteedBaseOffsetAlignment`

Type: [UINT](#)

Optional. Defines a minimum guaranteed alignment in bytes for the base offset of the buffer range that will contain this tensor, or 0 to provide no minimum guaranteed alignment. If specified, this value must be a power of two that is at least as large as the element size.

When binding this tensor, the offset in bytes of the buffer range from the start of the buffer must be a multiple of the *GuaranteedBaseOffsetAlignment*, if provided.

Buffer tensors always have a minimum alignment of 16 bytes. However, providing a larger value for the *GuaranteedBaseOffsetAlignment* may allow DirectML to achieve better performance, because a larger alignment enables the use of vectorized load/store instructions.

Although this member is optional, for best performance we recommend that you align tensors to boundaries of 32 bytes or more, where possible.

Requirements

Header		directml.h

See also

[Binding in DirectML](#)

DML_CAST_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that performs the cast function $f(x) = \text{cast}(x)$, casting each element in the input to the data type of the output tensor, and storing the result in the corresponding element in the output.

Syntax

```
struct DML_CAST_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_CONVOLUTION_DIRECTION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a direction for the DirectML convolution operator (as described by the [DML_CONVOLUTION_OPERATOR_DESC](#) structure).

Syntax

```
typedef enum DML_CONVOLUTION_DIRECTION {  
    DML_CONVOLUTION_DIRECTION_FORWARD,  
    DML_CONVOLUTION_DIRECTION_BACKWARD  
} ;
```

Constants

DML_CONVOLUTION_DIRECTION_FORWARD	Indicates a forward convolution.
DML_CONVOLUTION_DIRECTION_BACKWARD	Indicates a backward convolution. Backward convolution is also known as <i>transposed</i> convolution.

Requirements

Header	directml.h

See also

[DML_CONVOLUTION_OPERATOR_DESC](#)

DML_CONVOLUTION_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a mode for the DirectML convolution operator (as described by the [DML_CONVOLUTION_OPERATOR_DESC](#) structure).

Syntax

```
typedef enum DML_CONVOLUTION_MODE {
    DML_CONVOLUTION_MODE_CONVOLUTION,
    DML_CONVOLUTION_MODE_CROSS_CORRELATION
} ;
```

Constants

DML_CONVOLUTION_MODE_CONVOLUTION	Specifies the convolution mode.
DML_CONVOLUTION_MODE_CROSS_CORRELATION	Specifies the cross-correlation mode. If in doubt, use this mode—it is appropriate for the vast majority of machine learning (ML) models.

Requirements

Header	directml.h

See also

[DML_CONVOLUTION_OPERATOR_DESC](#)

DML_CONVOLUTION_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML matrix multiplication operator that performs a convolution function on the input, $\text{out}[j] = \text{x}[i]*\text{w}[0] + \text{x}[i+1]*\text{w}[1] + \text{x}[i+2]*\text{w}[2] + \dots + \text{x}[i+k]*\text{w}[k] + \text{bias}$.

Syntax

```
struct DML_CONVOLUTION_OPERATOR_DESC {
    const DML_TENSOR_DESC      *InputTensor;
    const DML_TENSOR_DESC      *FilterTensor;
    const DML_TENSOR_DESC      *BiasTensor;
    const DML_TENSOR_DESC      *OutputTensor;
    DML_CONVOLUTION_MODE       Mode;
    DML_CONVOLUTION_DIRECTION  Direction;
    UINT                      DimensionCount;
    const UINT                 *Strides;
    const UINT                 *Dilations;
    const UINT                 *StartPadding;
    const UINT                 *EndPadding;
    const UINT                 *OutputPadding;
    UINT                      GroupCount;
    const DML_OPERATOR_DESC    *FusedActivation;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

FilterTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the filter.

BiasTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the bias.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Mode

Type: **DML_CONVOLUTION_MODE**

The mode to use for the convolution operation. If in doubt, use

DML_CONVOLUTION_MODE_CROSS_CORRELATION—it is appropriate for the vast majority of machine

learning (ML) models.

`Direction`

Type: [DML_CONVOLUTION_DIRECTION](#)

The direction of the convolution operation.

`DimensionCount`

Type: [UINT](#)

The number of dimensions. This field determines the size of the *Strides*, *Dilations*, *StartPadding*, *EndPadding*, and *OutputPadding* arrays.

`Strides`

Type: `const UINT*`

A pointer to a constant array of [UINT](#) containing the lengths of the strides of the tensor.

`Dilations`

Type: `const UINT*`

A pointer to a constant array of [UINT](#) containing the value of the dilation along each axis of the filter.

`StartPadding`

Type: `const UINT*`

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

`EndPadding`

Type: `const UINT*`

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

`OutputPadding`

Type: `const UINT*`

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the output for the corresponding axis. Padding defaults to 0.

`GroupCount`

Type: [UINT](#)

The number of groups.

`FusedActivation`

Type: `const DML_OPERATOR_DESC*`

An optional pointer to a constant [DML_OPERATOR_DESC](#) containing the fused activation layer.

Requirements

Header	directml.h
--------	------------

DML_CREATE_DEVICE_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Supplies additional device creation options to [DMLCreateDevice](#). Values can be bitwise OR'd together.

Syntax

```
typedef enum DML_CREATE_DEVICE_FLAGS {  
    DML_CREATE_DEVICE_FLAG_NONE,  
    DML_CREATE_DEVICE_FLAG_DEBUG  
} ;
```

Constants

DML_CREATE_DEVICE_FLAG_NONE	No creation options are specified.
DML_CREATE_DEVICE_FLAG_DEBUG	Enables the DirectML debug layers. To use the debug layers, developer mode must be enabled, and the DirectML debug layers must be installed. If the DML_CREATE_DEVICE_FLAG_DEBUG flag is specified and either condition is not met, then DMLCreateDevice returns DXGI_ERROR_SDK_COMPONENT_MISSING .

Requirements

Header	directml.h
--------	------------

DML_DEPTH_TO_SPACE_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that rearranges (permutes) data from depth into blocks of spatial data. The operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the height and width dimensions.

This is the reverse transformation of [DML_SPACE_TO_DEPTH_OPERATOR_DESC](#).

Syntax

```
struct DML_DEPTH_TO_SPACE_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT BlockSize;  
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from. The input tensor's dimensions are [N,C,H,W], where N is the batch axis, C is the channel or depth, H is the height, and W is the width.

OutputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to. The output tensor's dimensions are [N, C / (BlockSize * BlockSize), H * BlockSize, W * BlockSize].

BlockSize

Type: [UINT](#)

Blocks of [BlockSize, BlockSize] are moved.

Requirements

Header	directml.h
--------	------------

See also

[DML_SPACE_TO_DEPTH_OPERATOR_DESC](#)

DML_DIAGONAL_MATRIX_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that generates an identity-like matrix with ones on the major diagonal and zeros everywhere else. The diagonal ones may be shifted (via *Offset*) where `output[i, i + Offset] = 1`, meaning that arguments of *Offset* greater than zero shifts all ones to the right, and less than zero shifts them to the left.

This generator operator is useful for models. The input is restricted to two dimensions because that's the primary use case.

```
output.shape = input.shape
output.type = if (exists(dtype)) then dtype else input.type
// Any axis greater than H and W are treated as batch counts.
for each coordinate in output tensor
    output[coordinate] = if (coordinate.h + k == coordinate.w) then 1 else 0
endfor

// Example.
// k = 0 // default identity matrix
// input[2, 3] = [...] // Only the shape matters; not the content.
// output = [[1, 0],
//           [0, 1],
//           [0, 0]]
// k = -1 // shift ones down
// input[2, 3] = [...] // Only the shape matters; not the content.
// output = [[0, 0],
//           [1, 0],
//           [0, 1]]
// k = -3 // Shift the diagonal line of ones so far as to make all zeroes.
// input[2, 3] = [...] // Only the shape matters; not the content.
// output = [[0, 0],
//           [0, 0],
//           [0, 0]]
// k = 1 // shift ones right
// input[3, 3] = [...] // Only the shape matters; not the content.
// output = [[0, 1, 0],
//           [0, 0, 1],
//           [0, 0, 0]]
```

Syntax

```
struct DML_DIAGONAL_MATRIX_OPERATOR_DESC {
    const DML_TENSOR_DESC *OutputTensor;
    INT                 Offset;
    FLOAT               Value;
};
```

Members

OutputTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

`Offset`

Type: [INT](#)

An offset to shift the ones by.

`Value`

Type: [FLOAT](#)

TBD

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_ABS_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise absolute value function $f(x) = \text{abs}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The absolute value of x is its magnitude without regard to its sign.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ABS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header

directml.h

DML_ELEMENT_WISE_ACOS_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise arccosine function $f(x) = \text{acos}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ACOS_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic cosine function $f(x) = \log(x + \sqrt{x * x - 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_ADD_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the function of adding every element in *ATensor* to its corresponding element in *BTensor*, $f(a, b) = a + b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_ADD_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_ADD1_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the function of adding every element in *ATensor* to its corresponding element in *BTensor*, $f(a, b) = a + b$, with the option for fused activation.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_ADD1_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

FusedActivation

Type: **const DML_OPERATOR_DESC***

An optional pointer to a constant **DML_OPERATOR_DESC** containing the fused activation layer.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)

Header	directml.h
---------------	------------

DML_ELEMENT_WISE_ASIN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise arcsine function $f(x) = \text{asin}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ASIN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_ASINH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic sine function $f(x) = \log(x + \sqrt{x * x + 1}) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ASINH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_ATAN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise arctangent function $f(x) = \text{atan}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ATAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_ATANH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = (\log((1 + x) / (1 - x)) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_ATANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Requirement	Description
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_CEIL_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise ceiling function $f(x) = \text{ceil}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The ceiling of x is the smallest integer that is greater than or equal to x .

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_CEIL_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from.

OutputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

ScaleBias

Type: [const DML_SCALE_BIAS*](#)

An optional pointer to a constant [DML_SCALE_BIAS](#) containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	File
	directml.h

DML_ELEMENT_WISE_CLIP_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise clip function $f(x) = \text{clamp}(x * \text{scale} + \text{bias}, \text{minValue}, \text{maxValue})$, where the scale and bias terms are optional, and where $\text{clamp}(x) = \min(\maxValue, \max(\minValue, x))$. This operator clamps (or limits) every element in the input within the closed interval $[\text{min}, \text{max}]$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_CLIP_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    FLOAT Min;
    FLOAT Max;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Min

Type: **FLOAT**

The minimum value, below which the operator replaces the value with *Min*. This is referred to as minValue in the operator description above in order not to confuse it with the min function.

Max

Type: **FLOAT**

The maximum value, above which the operator replaces the value with *Max*. This is referred to as maxValue in the

operator description above in order not to confuse it with the max function.

Requirements

Header	directml.h

DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs the element-wise constant power function $f(x) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional. The power function raises every element in the input to the power of the exponent.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    FLOAT Exponent;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Exponent

Type: **FLOAT**

The exponent to which to raise the input.

Requirements

Header

directml.h

See also

[DML_ELEMENT_WISE_POW_OPERATOR_DESC<](#)

DML_ELEMENT_WISE_COS_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise cosine function $f(x) = \cos(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_COS_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_COSH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise hyperbolic cosine function $f(x) = ((e^x + e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_COSH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Requirement	Description
Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs the linear dequantize function on every element in *InputTensor* with respect to its corresponding element in *ScaleTensor* and *ZeroPointTensor*, $f(\text{input}, \text{scale}, \text{zero_point}) = (\text{input} - \text{zero_point}) * \text{scale}$. Quantizing involves converting to a lower-precision data type in order to accelerate arithmetic.

Syntax

```
struct DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *ZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

ScaleTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing scale.

ZeroPointTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the zero point.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

See also

[DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC](#)

DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the function of dividing every element in *ATensor* by its corresponding element in *BTensor*, $f(a, b) = a / b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

DML_ELEMENT_WISE_ERF_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an elementwise Erf error function $\text{erf}(x) = 2 / \sqrt{\pi} * \int_{0}^{|x|} e^{-t^2} dt$ on the input.

```
For each x in InputTensor
{
    // Constants to approximate function via polynomial.
    a1 = 0.254829592
    a2 = -0.284496736
    a3 = 1.421413741
    a4 = -1.453152027
    a5 = 1.061405429
    p = 0.3275911

    // Save the sign of x.
    int sign = if (x < 0) then -1 else 1
    x = fabs(x)

    // A&S formula 7.1.26.
    t = 1.0 / (1.0 + p*x)
    y = 1.0 - (((((a5*t + a4)*t) + a3)*t + a2)*t + a1)*t*exp(-x*x)
    x = sign * y
}
```

Syntax

```
struct DML_ELEMENT_WISE_ERF_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this

has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_EXP_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise natural exponential function $f(x) = \exp(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_EXP_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise floor function $f(x) = \text{floor}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional. The floor of x is the largest integer that is less than or equal to x .

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML generic operator that performs the element-wise identity function $f(x) = x * \text{scale} + \text{bias}$. The operator effectively copies its input tensor to the output, while applying optional scale and bias terms. The data types and sizes of the input and output tensors must be the same.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to. This operator supports in-place execution. That is, the supplied output tensor may be the same as the supplied input tensor.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_IF_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that essentially performs a ternary `if` statement. It evaluates every element in *ConditionTensor*, and if `true` returns the corresponding element in *ATensor*, otherwise it returns the corresponding element in *BTensor*. Each input tensor can broadcast to the output shape.

```
For each condition, a, b in ConditionTensor, ATensor, BTensor
  if (b) then x else y

// Example.
// condition = bool [[1, 0], [1, 1]]
// x = [[1, 2], [3, 4]] // first input
// y = [[9, 8], [7, 6]] // second input
// z = [[1, 8], [3, 4]] // output
```

Can be used to functionally build up other aggregate operators, such as LeakyRelu. Here's an illustration in pseudo-code (not the most efficiently, but possible): `LeakyRelu(X) = If(Less(X, 0), Mul(X, alpha), X)`.

Syntax

```
struct DML_ELEMENT_WISE_IF_OPERATOR_DESC {
    const DML_TENSOR_DESC *ConditionTensor;
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`ConditionTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the *Condition* tensor to read from.

`ATensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the *A* tensor to read from.

`BTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the *B* tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that determines, elementwise, whether the input is NaN.

```
For each x in InputTensor
    // Can test float32 IEEE 754 NaN by looking for an 8-bit exponent set to all ones
    // and any bit of the 23-bit mantissa set (sign-ignorable).
    IsNaN(x) = (asuint(x) & 0x7FFFFFFF) > 0x7F800000
```

Syntax

```
struct DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_LOG_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise natural logarithm function $f(x) = \log(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_LOG_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical AND function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = a \&\& b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical equality function between every element in $ATensor$ and its corresponding element in $BTensor$, $f(a, b) = (a == b)$.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the A tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the B tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical greater-than function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = (a > b)$.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical less-than function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = (a < b)$.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
---------------	------------

DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical NOT function on every element in the input, $f(x) = !x$.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header

directml.h

DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC structure

5/12/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical OR function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = a \parallel b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a logical exclusive OR (XOR) function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = a \text{ xor } b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

DML_ELEMENT_WISE_MAX_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math reduction operator that performs a maximum function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = \max(a, b)$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_MAX_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h

DML_ELEMENT_WISE_MEAN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math reduction operator that performs an arithmetic mean function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = (a + b) / 2$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_MEAN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_MIN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math reduction operator that performs a minimum function between every element in *ATensor* and its corresponding element in *BTensor*, $f(a, b) = \min(a, b)$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_MIN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the function of multiplying every element in *ATensor* by its corresponding element in *BTensor*, $f(a, b) = a * b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_POW_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise power function $f(x, \text{exponent}) = \text{pow}(x * \text{scale} + \text{bias}, \text{exponent})$, where the scale and bias terms are optional. The power function raises every element in the input to the power of its corresponding element in the exponent.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_POW_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *ExponentTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

ExponentTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the exponent to which to raise the input.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header

directml.h

See also

[DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC](#)

DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs the linear quantize function on every element in *InputTensor* with respect to its corresponding element in *ScaleTensor* and *ZeroPointTensor*, $f(\text{input}, \text{scale}, \text{zero_point}) = \text{clamp}(\text{round}(\text{input} / \text{scale}) + \text{zero_point}, 0, 255)$. Quantizing involves converting to a lower-precision data type in order to accelerate arithmetic.

Syntax

```
struct DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *ZeroPointTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

ScaleTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing scale.

ZeroPointTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the zero point.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

See also

[DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC](#)

DML_ELEMENT_WISE_RECIP_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a reciprocal function on every element in the input, $f(x) = 1 / (x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_RECIP_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_SIGN_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that, elementwise, returns the sign of the input.

```
For each x in InputTensor
    if (x > 0) then 1
    else if (x < 0) then -1
    else if (x == 0) then 0
    else NaN
```

Syntax

```
struct DML_ELEMENT_WISE_SIGN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_SIN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise sine function $f(x) = \sin(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_SIN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_SINH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise hyperbolic sine function $f(x) = ((e^x - e^{-x}) / 2) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_SINH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_SQRT_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs a square root function on every element in the input, $f(x) = \text{sqrt}(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_SQRT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the function of subtracting every element in *BTensor* from its corresponding element in *ATensor*, $f(a, b) = a - b$.

This operator supports in-place execution, meaning the output tensor is permitted to alias one of the input tensors during binding.

Syntax

```
struct DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC {  
    const DML_TENSOR_DESC *ATensor;  
    const DML_TENSOR_DESC *BTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Requirements

Header	directml.h
--------	------------

DML_ELEMENT_WISE_TAN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise tangent function $f(x) = \tan(x * \text{scale} + \text{bias})$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_TAN_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_SCALE_BIAS *ScaleBias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Header	directml.h

DML_ELEMENT_WISE_TANH_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML trigonometric operator that performs the element-wise inverse hyperbolic tangent function $f(x) = \tanh(x) * \text{scale} + \text{bias}$, where the scale and bias terms are optional.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_TANH_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleBias

Type: **const DML_SCALE_BIAS***

An optional pointer to a constant **DML_SCALE_BIAS** containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML math operator that performs the element-wise threshold function $f(x) = \max(x * \text{scale} + \text{bias}, \text{min})$, where the scale and bias terms are optional. The threshold of x with respect to min is the larger of the two values.

This operator supports in-place execution, meaning the output tensor is permitted to alias the input tensor during binding.

Syntax

```
struct DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    const DML_SCALE_BIAS *ScaleBias;
    FLOAT Min;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

`ScaleBias`

Type: `const DML_SCALE_BIAS*`

An optional pointer to a constant `DML_SCALE_BIAS` containing scale and bias to apply to the input. If present, this has the effect of applying the function $g(x) = x * \text{scale} + \text{bias}$ to each element before this topic's operator is applied.

`Min`

Type: `FLOAT`

The minimum value, below which the operator replaces the value with `Min`.

Requirements

Header

`directml.h`

DML_EXECUTION_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Supplies options to DirectML to control execution of operators. These flags can be bitwise OR'd together to specify multiple flags at once.

Syntax

```
typedef enum DML_EXECUTION_FLAGS {
    DML_EXECUTION_FLAG_NONE,
    DML_EXECUTION_FLAG_ALLOW_HALF_PRECISION_COMPUTATION,
    DML_EXECUTION_FLAG_DISABLE_META_COMMANDS,
    DML_EXECUTION_FLAG_DESCRIPTOR_VOLATILE
} ;
```

Constants

DML_EXECUTION_FLAG_NONE	No execution flags are specified.
DML_EXECUTION_FLAG_ALLOW_HALF_PRECISION_COMPUTATION	Allows DirectML to perform computation using half-precision floating-point (FP16), if supported by the hardware device.
DML_EXECUTION_FLAG_DISABLE_META_COMMANDS	Forces DirectML execute the operator using DirectCompute instead of meta commands. DirectML uses meta commands by default, if available.
DML_EXECUTION_FLAG_DESCRIPTOR_VOLATILE	<p>Allows changes to bindings after an operator's execution has been recorded in a command list, but before it has been submitted to the command queue. By default, without this flag set, you must set all bindings on the binding table before you record an operator into a command list.</p> <p>This flag allows you to perform late binding—that is, to set (or to change) bindings on operators that you've already recorded into a command list. However, this may result in a performance penalty on some hardware, as it prohibits drivers from promoting static descriptor accesses to root descriptor accesses.</p> <p>For more info, see DESCRIPTORS_VOLATILE.</p>

Requirements

Header	directml.h
--------	------------

See also

[Binding in DirectML](#)

DML_FEATURE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines a set of optional features and capabilities that can be queried from the DirectML device. See [IDMLDevice::CheckFeatureSupport](#).

Syntax

```
typedef enum DML_FEATURE {
    DML_FEATURE_TENSOR_DATA_TYPE_SUPPORT,
    DML_FEATURE_FEATURE_LEVELS
} ;
```

Constants

DML_FEATURE_TENSOR_DATA_TYPE_SUPPORT	Allows querying for tensor data type support. The query type is DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT , and the support data type is DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT .
--------------------------------------	---

Requirements

Header	directml.h
--------	------------

See also

[IDMLDevice::CheckFeatureSupport](#)

DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Provides detail about whether a DirectML device supports a particular data type within tensors. See [IDMLDevice::CheckFeatureSupport](#). The query type is [DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT](#), and the support data type is [DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT](#).

Syntax

```
struct DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT {  
    BOOL IsSupported;  
};
```

Members

`IsSupported`

Type: [BOOL](#)

TRUE if the tensor data type is supported within tensors by the DirectML device. Otherwise, FALSE.

Requirements

Header	directml.h
--------	------------

See also

[IDMLDevice::CheckFeatureSupport](#)

DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Used to query a DirectML device for its support for a particular data type within tensors. See [IDMLDevice::CheckFeatureSupport](#). The query type is **DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT**, and the support data type is [DML_FEATURE_DATA_TENSOR_DATA_TYPE_SUPPORT](#).

Syntax

```
struct DML_FEATURE_QUERY_TENSOR_DATA_TYPE_SUPPORT {
    DML_TENSOR_DATA_TYPE DataType;
};
```

Members

`DataType`

Type: [DML_TENSOR_DATA_TYPE](#)

The data type about which you're querying for support.

Requirements

<code>Header</code>	<code>directml.h</code>

DML_GATHER_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator which, when given a data tensor of rank $r \geq 1$, and an indices tensor of rank q , gathers the entries in the axis dimension of the data (by default, the outermost one is $\text{axis} == 0$) indexed by indices, and concatenates them in an output tensor of rank $q + (r - 1)$.

Syntax

```
struct DML_GATHER_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
    UINT                 IndexDimensions;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the indices.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

`Axis`

Type: `UINT`

The axis dimension of the input tensor to gather on.

`IndexDimensions`

Type: `UINT`

The number of index dimensions.

Requirements

Header	directml.h
--------	------------

DML_GEMM_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a general matrix multiplication function on the input, $y = \text{alpha} * \text{transposeA}(A) * \text{transposeB}(B) + \text{beta} * C$.

Syntax

```
struct DML_GEMM_OPERATOR_DESC {
    const DML_TENSOR_DESC *ATensor;
    const DML_TENSOR_DESC *BTensor;
    const DML_TENSOR_DESC *CTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_MATRIX_TRANSFORM TransA;
    DML_MATRIX_TRANSFORM TransB;
    FLOAT Alpha;
    FLOAT Beta;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

ATensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *A* tensor to read from. This tensor's dimensions should be [M, K] if *TransA* is **DML_MATRIX_TRANSFORM_NONE**, or [K, M] if *TransA* is **DML_MATRIX_TRANSFORM_TRANSPOSE**.

BTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the *B* tensor to read from. This tensor's dimensions should be [K, N] if *TransB* is **DML_MATRIX_TRANSFORM_NONE**, or [N, K] if *TransB* is **DML_MATRIX_TRANSFORM_TRANSPOSE**.

CTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the description of the *C* tensor to read from, or **nullptr**. This tensor's dimensions should be unidirectional broadcastable to [M, N].

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to. This tensor's dimensions are [M, N].

TransA

Type: **DML_MATRIX_TRANSFORM**

The transform to be applied to *ATensor*; either a transpose, or no transform.

TransB

Type: [DML_MATRIX_TRANSFORM](#)

The transform to be applied to *BTensor*; either a transpose, or no transform.

Alpha

Type: [FLOAT](#)

The value of the scalar multiplier for the product of inputs *ATensor* and *BTensor*.

Beta

Type: [FLOAT](#)

The value of the scalar multiplier for the optional input *CTensor*.

FusedActivation

Type: [const DML_OPERATOR_DESC*](#)

An optional pointer to a constant [DML_OPERATOR_DESC](#) containing the fused activation layer.

Requirements

Header	File
	directml.h

DML_GRU_OPERATOR_DESC structure

5/13/2020 • 4 minutes to read • [Edit Online](#)

Describes a DirectML deep learning operator that performs a (standard layers) one-layer gated recurrent unit (GRU) function on the input. This operator uses multiple gates to perform this layer. These gates are performed multiple times in a loop dictated by the sequence length dimension and the *SequenceLengthsTensor* argument.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)
{
    $z_t = f(X_t*W_z^T + H_{t-1} * R_z^T + W_{bz} + R_{bz})$
    $r_t = f(X_t*W_r^T + H_{t-1} * R_r^T + W_{br} + R_{br})$

    if (LinearBeforeReset = 0)
        $h_t = g(X_t*W_h^T + (r_t \bigodot H_{t-1}) * R_h^T + W_bh + R_bh)$
    else
        $h_t = g(X_t*W_h^T + (r_t \bigodot (H_{t-1} * R_h^T + R_bh) + W_bh))$

    $H_t = ((1 - z_t) \bigodot h_t) + (z_t \bigodot H_{t-1})$
}
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)
{
    $z_t = f(X_t*W_{Bz}^T + H_{t-1} * R_{Bz}^T + W_{Bbz} + R_{Bbz})$
    $r_t = f(X_t*W_{Br}^T + H_{t-1} * R_{Br}^T + W_{Bbr} + R_{Bbr})$

    if (LinearBeforeReset = 0)
        $h_t = g(X_t*W_{Bh}^T + (r_t \bigodot H_{t-1}) * R_{Bh}^T + W_{Bbh} + R_{Bbh})$
    else
        $h_t = g(X_t*W_{Bh}^T + (r_t \bigodot (H_{t-1} * R_{Bh}^T + R_{Bbh}) + W_{Bbh}))$

    $H_t = ((1 - z_t) \bigodot h_t) + (z_t \bigodot H_{t-1})$
}
```

Equation legend

- \$t\$ current GRU layer index.
- \$t-1\$ previous GRU layer index. If backwards direction, then it means \$t+1\$ index but still the previously calculated \$t\$.
- \$T\$ signifies that a matrix will be transposed before use.
- \$*\$ signifies a matrix multiplication.
- \$+\$ signifies an element-wise addition of two matrices.
- \$\bigodot\$ signifies an element-wise multiply of two matrices.
- \$H_t\$ output of current GRU index \$t\$.
- \$f(),g()\$ activation functions dictated by *ActivationDescs*.
- \$X_t\$ Sub-matrix of the *InputTensor*, which is defined by matrix size [1, batch_size, input_size], at index \$t\$ of the seq_length dimension.
- \$W_{[zrh]}^T\$ Forward sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- \$W_{[Bzrh]}^T\$ Backwards sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size,

`input_size]`. This matrix will be transposed before use.

- `$H_{t-1}` is an intermediate tensor which is defined to be the output of the previous layer. For the first index of `$t`, `H_{t-1}` is replaced with `HiddenInitTensor`. This is defined to be size [1, batch_size, hidden_size]. A different `HiddenInitTensor` sub-matrix is used for each direction.
- `$R_{[zrh]}^T` Forward sub-matrix of `RecurrenceTensor`, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- `$R_{B[zrh]}^T` Backwards sub-matrix of `RecurrenceTensor`, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- `$W_{b[zrh]}`, `$R_{b[zrh]}` are the bias tensors of the weight tensor, and recurrence tensors, which are sub-matrices of `BiasTensor`. This matrix is size [1,hidden_size] and is broadcasted up to the required size which is [1, batch_size, hidden_size].
- `$W_{Bb[zrh]}`, `$R_{Bb[zrh]}` are the bias tensors for the backwards direction
- `$z_t` stands for update gate at index `t`.
- `$r_t` stands for reset gate at index `t`.
- `$h_t` stands for hidden gate at index `t`.

Syntax

```
struct DML_GRU_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *WeightTensor;
    const DML_TENSOR_DESC *RecurrenceTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *HiddenInitTensor;
    const DML_TENSOR_DESC *SequenceLengthsTensor;
    const DML_TENSOR_DESC *OutputSequenceTensor;
    const DML_TENSOR_DESC *OutputSingleTensor;
    UINT ActivationDescCount;
    const DML_OPERATOR_DESC *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
    BOOL LinearBeforeReset;
};
```

Members

InputTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from for the input tensor, `X`. Packed (and potentially padded) into one 4-D tensor with the shape of [1, seq_length, batch_size, input_size]. seq_length is the dimension that is mapped to the GRU index, `t`.

WeightTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the weight tensor, `W`. Concatenation of `$W_{[zrh]}` and `$W_{B[zrh]}` (if bidirectional). The tensor has shape [1, num_directions, 3 * hidden_size, input_size].

RecurrenceTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the recurrence tensor, `R`. Concatenation of `$R_{[zrh]}` and `$R_{B[zrh]}` (if bidirectional). The tensor has shape [1, num_directions, 3 * hidden_size, hidden_size].

`BiasTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the bias tensor, $\$B\$$. Concatenation of $(\$W_{\{b[zrh]\}}, \$R_{\{b[zrh]\}})$ and $(\$W_{\{Bb[zrh]\}}, \$R_{\{Bb[zrh]\}})$ (if bidirectional). The tensor has shape [1, 1, num_directions, 6 * hidden_size].

`HiddenInitTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the hidden node initializer tensor, $\$H_{\{t-1\}}$ for the first loop index $\$t$$. If not specified, then defaults to 0. This tensor has shape [1, num_directions, batch_size, hidden_size].

`SequenceLengthsTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing an independent seq_length for each element in the batch. If not specified, then all sequences in the batch have length seq_length. This tensor has shape [1, 1, 1, batch_size].

`OutputSequenceTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the concatenation of all the intermediate output values of the hidden nodes, $\$H_t$. This tensor has shape [seq_length, num_directions, batch_size, hidden_size]. seq_length is mapped to the loop index $\$t$$.

`OutputSingleTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the last output value of the hidden nodes, $\$H_t$. This tensor has shape [1, num_directions, batch_size, hidden_size].

`ActivationDescCount`

Type: `UINT`

This field determines the size of the *ActivationDescs* array.

`ActivationDescs`

Type: `const DML_OPERATOR_DESC*`

A pointer to a constant array of `DML_OPERATOR_DESC` containing the descriptions of the activation operators, $\$f()\$$ and $\$g()\$$. Both $\$f()\$$ and $\$g()\$$ are defined independently of direction, meaning that if `DML_RECURRENT_NETWORK_DIRECTION_FORWARD` or `DML_RECURRENT_NETWORK_DIRECTION_BACKWARD` are defined 2 activations must be provided. If `DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL` is defined 4 activations must be provided. For bidirectional, activations must be provided $\$f()\$$ and $\$g()\$$ for forward followed by $\$f()\$$ and $\$g()\$$ for backwards.

`Direction`

Type: `const DML_RECURRENT_NETWORK_DIRECTION*`

The direction of the operator—forward, backwards, or bidirectional.

LinearBeforeReset

Type: **BOOL**

TRUE to specify that, when computing the output of the hidden gate, the linear transformation should be applied before multiplying by the output of the reset gate. Otherwise, FALSE.

Requirements

Header	
	directml.h

DML_INTERPOLATION_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a mode for the DirectML upsample 2-D operator (as described by the [DML_UPSAMPLE_2D_OPERATOR_DESC](#) structure).

Syntax

```
typedef enum DML_INTERPOLATION_MODE {  
    DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR,  
    DML_INTERPOLATION_MODE_LINEAR  
} ;
```

Constants

DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR	Specifies the nearest-neighbor mode.
DML_INTERPOLATION_MODE_LINEAR	Specifies a linear (including bilinear, trilinear, etc.) mode.

Requirements

Header	directml.h

See also

[DML_UPSAMPLE_2D_OPERATOR_DESC](#)

DML_JOIN_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a join function on an array of input tensors.

Syntax

```
struct DML_JOIN_OPERATOR_DESC {  
    UINT             InputCount;  
    const DML_TENSOR_DESC *InputTensors;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT             Axis;  
};
```

Members

InputCount

Type: **UINT**

This field determines the size of the *InputTensors* array.

InputTensors

Type: **const DML_TENSOR_DESC***

A pointer to a constant array of **DML_TENSOR_DESC** containing the descriptions of the tensors to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Axis

Type: **UINT**

The axis dimension of the input tensor to join on.

Requirements

Header

directml.h

DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a local response normalization (LRN) function on the input, $y = x / (\text{bias} + (\alpha / \text{size}) * \sum(x_i^2 \text{ for every } x_i \text{ in the local region}))^\beta$. The data type and size of the input and output tensors must be the same.

Syntax

```
struct DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    BOOL CrossChannel;
    UINT LocalSize;
    FLOAT Alpha;
    FLOAT Beta;
    FLOAT Bias;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from. This tensor's dimensions for the image case are [N, C, H, W], where N is the batch size, C is the number of channels, and H and W are the height and the width of the data. For the non-image case, the dimensions are in the form of [N, C, D1, D2, ..., Dn], where N is the batch size.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to. This tensor's dimensions match those of `InputTensor`.

`CrossChannel`

Type: `BOOL`

TRUE if the LRN layer is channel-wise (cross-channel). Otherwise, FALSE.

`LocalSize`

Type: `UINT`

The number of channels to sum over.

`Alpha`

Type: `FLOAT`

The value of the scaling parameter. You can use a default value of 0.0001.

`Beta`

Type: `FLOAT`

The value of the exponent. You can use a default value of 0.75.

`Bias`

Type: `FLOAT`

The value of bias. You can use a default value of 1.0.

Remarks

The operator normalizes over local input regions. The local region is defined across the channels. For an element $X[n, c, d1, \dots, dk]$ in a tensor of shape $(N \times C \times D1 \times D2, \dots, Dk)$, its region is $\{X[n, i, d1, \dots, dk] \mid \max(0, c - \text{floor}((\text{size} - 1) / 2)) \leq i \leq \min(C - 1, c + \text{ceil}((\text{size} - 1) / 2))\}$.

$\text{square_sum}[n, c, d1, \dots, dk] = \sum(X[n, i, d1, \dots, dk]^2)$, where $\max(0, c - \text{floor}((\text{size} - 1) / 2)) \leq i \leq \min(C - 1, c + \text{ceil}((\text{size} - 1) / 2))$.

$Y[n, c, d1, \dots, dk] = X[n, c, d1, \dots, dk] / (\text{bias} + \alpha / \text{size} * \text{square_sum}[n, c, d1, \dots, dk])^{\beta}$.

Requirements

Header	
	directml.h

DML_LP_NORMALIZATION_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an L_p-normalization function along the specified axis of the input tensor.

Syntax

```
struct DML_LP_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
    FLOAT                Epsilon;
    UINT                 P;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Axis

Type: **UINT**

The axis on which to apply normalization. You can use a default value of -1 to mean the last axis.

Epsilon

Type: **FLOAT**

The epsilon value to use to avoid division by zero.

P

Type: **UINT**

The order of the normalization (either 1 or 2). You can use a default value of 2.

Requirements

Header	directml.h
--------	------------

DML_LP_POOLING_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an L_p pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = (x_1^p + x_2^p + \dots + x_n^p)^{(1/p)}$ where $X \rightarrow Y$ reduced for each kernel.

L_p pooling consists of computing the L_p norm on all values of a subset of the input tensor according to the kernel size, and then downsampling the data into the output tensor Y for further processing.

Syntax

```
struct DML_LP_POOLING_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT DimensionCount;  
    const UINT *Strides;  
    const UINT *WindowSize;  
    const UINT *StartPadding;  
    const UINT *EndPadding;  
    UINT P;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

DimensionCount

Type: **UINT**

The number of dimensions. This field determines the size of the *Strides*, *WindowSize*, *StartPadding*, and *EndPadding* arrays.

Strides

Type: **const UINT***

A pointer to a constant array of **UINT** containing the lengths of the strides of the tensor.

WindowSize

Type: **const UINT***

A pointer to a constant array of **UINT** containing the lengths of the pooling windows.

StartPadding

Type: **const [UINT](#)***

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

`EndPadding`

Type: **const [UINT](#)***

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

`P`

Type: [UINT](#)

The p value of the L_p norm used to pool over the input data. You can use a default value of 2.

Requirements

Header	<code>directml.h</code>
---------------	-------------------------

DML_LSTM_OPERATOR_DESC structure

5/13/2020 • 6 minutes to read • [Edit Online](#)

Describes a DirectML deep learning operator that performs a one-layer long short term memory (LSTM) function on the input. This operator uses multiple gates to perform this layer. These gates are performed multiple times in a loop, dictated by the sequence length dimension and the *SequenceLengthsTensor* argument.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)
{
    $i_t = f(clip(X_t*W_i^T + H_{t-1} * R_i^T + P_i \bigodot C_{t-1} + W_{bi} + R_{bi}))$

    if (CoupleInputAndForget = 0)
        $f_t = f(clip(X_t*W_f^T + H_{t-1} * R_f^T + P_f \bigodot C_{t-1} + W_{bf} + R_{bf}))$
    else
        $f_t = 1.0 - i_t$

    $c_t = g(clip(X_t*W_c^T + H_{t-1} * R_c^T + W_{bc} + R_{bc}))$
    $C_t = f_t \bigodot C_{t-1} + i_t \bigodot c_t
    $o_t = f(clip(X_t*W_o^T + H_{t-1} * R_o^T + P_o \bigodot C_{t-1} + W_{bo} + R_{bo}))$
    $H_t = o_t \bigodot h(ct)$
}
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)
{
    $i_t = f(clip(X_t*W_{Bi}^T + H_{t-1} * R_{Bi}^T + P_i \bigodot C_{t-1} + W_{Bbi} + R_{Bbi}))$

    if (CoupleInputAndForget = 0)
        $f_t = f(clip(X_t*W_{Bf}^T + H_{t-1} * R_{Bf}^T + P_f \bigodot C_{t-1} + W_{Bbf} + R_{Bbf}))$
    else
        $f_t = 1.0 - i_t$

    $c_t = g(clip(X_t*W_{Bc}^T + H_{t-1} * R_{Bc}^T + W_{BBC} + R_{BBC}))$
    $C_t = f_t \bigodot C_{t-1} + i_t \bigodot c_t
    $o_t = f(clip(X_t*W_{Bo}^T + H_{t-1} * R_{Bo}^T + P_o \bigodot C_{t-1} + W_{Bbo} + R_{Bbo}))$
    $H_t = o_t \bigodot h(ct)$
}
```

Equation legend

- t current LSTM layer index.
- $t-1$ previous LSTM layer index. If backwards direction, then it means $t+1$ index but still the previously calculated t .
- T signifies that a matrix will be transposed before use.
- $* \bigodot$ signifies a matrix multiplication.
- $+ \bigoplus$ signifies an element-wise addition of two matrices.
- \bigodot signifies an element-wise multiply of two matrices.
- $f(), g(), h()$ activation functions dictated by *ActivationDescs*.
- $clip()$ optional clipping operation, which is controlled by *UseClipThreshold*.
- X_t Sub-matrix of the *InputTensor*, which is defined by matrix size [1, batch_size, input_size], at index t of the seq_length dimension.

- $\$W_{\{iofc\}}^T$ Forward sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- $\$W_{\{B[iofc]\}}^T$ Backwards sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- $\$R_{\{iofc\}}^T$ Forward sub-matrix of *RecurrenceTensor*, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- $\$R_{\{B[iofc]\}}^T$ Backwards sub-matrix of *RecurrenceTensor*, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- $\$H_t$ output of current LSTM index t .
- $\$H_{t-1}$ an intermediate tensor that's defined to be the output of the previous layer. For the first index of t , $\$H_{t-1}$ is replaced with *HiddenInitTensor*. This is defined to be size [1, batch_size, hidden_size]. A different *HiddenInitTensor* sub-matrix is used for each direction.
- $\$C_t$ is the tensor containing the Cell Memory for index t .
- $\$C_{t-1}$ an intermediate tensor that's defined to be the Cell Memory of the previous layer. For the first index of t , $\$C_{t-1}$ is replaced with *CellMemoryInitTensor*. This is defined to be size [1, batch_size, hidden_size]. A different *CellMemoryInitTensor* sub-matrix is used for each direction.
- $\$W_{\{b[iofc]\}}$, $\$R_{\{b[iofc]\}}$ are the bias tensors for the weight tensor, and recurrence tensor, which are sub-matrices of *BiasTensor*. These matrices are size [1,hidden_size], and are broadcasted up to the required size, which is [1, batch_size, hidden_size].
- $\$W_{\{Bb[iofc]\}}$, $\$R_{\{Bb[iofc]\}}$ are the bias tensors for the backwards direction.
- $\$i_t$ stands for input gate at index t .
- $\$o_t$ stands for output gate at index t .
- $\$f_t$ stands for forget gate at index t .
- $\$c_t$ stands for cell gate at index t .

Syntax

```
struct DML_LSTM_OPERATOR_DESC {
    const DML_TENSOR_DESC          *InputTensor;
    const DML_TENSOR_DESC          *WeightTensor;
    const DML_TENSOR_DESC          *RecurrenceTensor;
    const DML_TENSOR_DESC          *BiasTensor;
    const DML_TENSOR_DESC          *HiddenInitTensor;
    const DML_TENSOR_DESC          *CellMemInitTensor;
    const DML_TENSOR_DESC          *SequenceLengthsTensor;
    const DML_TENSOR_DESC          *PeepholeTensor;
    const DML_TENSOR_DESC          *OutputSequenceTensor;
    const DML_TENSOR_DESC          *OutputSingleTensor;
    const DML_TENSOR_DESC          *OutputCellSingleTensor;
    UINT                           ActivationDescCount;
    const DML_OPERATOR_DESC        *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
    float                          ClipThreshold;
    BOOL                           UseClipThreshold;
    BOOL                           CoupleInputForget;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from, X . Packed (and potentially padded) into one 4-D tensor with the shape of [1, seq_length, batch_size, input_size]. seq_length is the

dimension that is mapped to the GRU index, t .

WeightTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the weight tensor for the gates, W . Concatenation of $W_{[iofc]}$ and $W_{B[iofc]}$ (if bidirectional). The tensor has shape [1, num_directions, 4 * hidden_size, input_size].

RecurrenceTensor

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the recurrence weight tensor, R . Concatenation of $R_{[iofc]}$ and $R_{B[iofc]}$ (if bidirectional). This tensor has shape [1, num_directions, 4 * hidden_size, hidden_size].

BiasTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the bias tensor for the input gate, B . Concatenation of $W_{b[iofc]}$, $R_{b[iofc]}$, and $W_{Bb[iofc]}$, $R_{Bb[iofc]}$ (if bidirectional). This tensor has shape [1, 1, num_directions, 8 * hidden_size]. If not specified, then defaults to 0 bias.

HiddenInitTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the hidden node initializer tensor, H_{t-1} . Contents of this tensor are only used on the first loop index t . If not specified, then defaults to 0. This tensor has shape [1, num_directions, batch_size, hidden_size].

CellMemInitTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the cell initializer tensor, C_{t-1} . Contents of this tensor are only used on the first loop index t . If not specified, then defaults to 0. This tensor has shape [1, num_directions, batch_size, hidden_size].

SequenceLengthsTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the lengths of the sequences in a batch. If not specified, then all sequences in the batch have length seq_length. This tensor has shape [1, 1, 1, batch_size].

PeepholeTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the weight tensor for peepholes, P . If not specified, then defaults to 0. Concatenation of $P_{[iof]}$ and $P_{B[iof]}$ (if bidirectional). This tensor has shape [1, 1, num_directions, 3 * hidden_size].

OutputSequenceTensor

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the

concatenation of all the intermediate output values of the hidden nodes, H_t . This tensor has shape [seq_length, num_directions, batch_size, hidden_size]. seq_length is mapped to the loop index t .

`OutputSingleTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the last output value of the hidden nodes, H_t . This tensor has shape [1, num_directions, batch_size, hidden_size].

`OutputCellSingleTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the last output value of the cell, C_t . This tensor has shape [1, num_directions, batch_size, hidden_size].

`ActivationDescCount`

Type: `UINT`

This field determines the size of the *ActivationDescs* array.

`ActivationDescs`

Type: `const DML_OPERATOR_DESC*`

A pointer to a constant array of `DML_OPERATOR_DESC` containing the descriptions of the activation operators $f()$, $g()$, and $h()$. $f()$, $g()$, and $h()$ are defined independently of direction, meaning that if `DML_RECURRENT_NETWORK_DIRECTION_FORWARD` or `DML_RECURRENT_NETWORK_DIRECTION_BACKWARD` is defined, then 3 activations must be provided. If `DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL` is defined, then 6 activations must be provided. For bidirectional, activations must be provided $f()$, $g()$, and $h()$ for forward followed by $f()$, $g()$, and $h()$ for backwards.

`Direction`

Type: `const DML_RECURRENT_NETWORK_DIRECTION*`

The direction of the operator—forward, reverse, or bidirectional. You can use a default value of `DML_RECURRENT_NETWORK_DIRECTION_FORWARD`.

`ClipThreshold`

Type: `float`

The cell clip threshold. Clipping bounds the elements of a tensor in the range of [-threshold, +threshold], and is applied to the input of activations.

`UseClipThreshold`

Type: `BOOL`

TRUE if *ClipThreshold* should be used. Otherwise, FALSE.

`CoupleInputForget`

Type: `BOOL`

TRUE if the input and forget gates should be coupled. Otherwise, FALSE.

Requirements

Header	directml.h
--------	------------

DML_MATRIX_TRANSFORM enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a matrix transform to be applied to a DirectML tensor.

Syntax

```
typedef enum DML_MATRIX_TRANSFORM {  
    DML_MATRIX_TRANSFORM_NONE,  
    DML_MATRIX_TRANSFORM_TRANSPOSE  
} ;
```

Constants

DML_MATRIX_TRANSFORM_NONE	Specifies that no transform is to be applied.
DML_MATRIX_TRANSFORM_TRANSPOSE	Specifies that a transpose transform is to be applied.

Requirements

Header	directml.h
--------	------------

DML_MAX_POOLING_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = \max(x_1 + x_2 + \dots + x_{\text{pool_size}})$.

Max pooling consists of computing the max on all values of a subset of the input tensor according to the kernel size, and then downsampling the data into the output tensor Y for further processing.

[DML_MAX_POOLING1_OPERATOR_DESC](#) is an updated version of [DML_MAX_POOLING_OPERATOR_DESC](#).

Syntax

```
struct DML_MAX_POOLING_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT DimensionCount;  
    const UINT *Strides;  
    const UINT *WindowSize;  
    const UINT *StartPadding;  
    const UINT *EndPadding;  
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from. This tensor's dimensions for the image case are [N, C, H, W], where N is the batch size, C is the number of channels, and H and W are the height and the width of the data. For the non-image case, the dimensions are in the form of [N, C, D1, D2, ..., Dn], where N is the batch size.

OutputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

DimensionCount

Type: [UINT](#)

The number of dimensions. This field determines the size of the *Strides*, *WindowSize*, *StartPadding*, and *EndPadding* arrays.

Strides

Type: [const UINT*](#)

A pointer to a constant array of [UINT](#) containing the lengths of the strides of the tensor.

WindowSize

Type: [const UINT*](#)

A pointer to a constant array of **UINT** containing the lengths of the pooling windows.

StartPadding

Type: **const UINT***

A pointer to a constant array of **UINT** containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

EndPadding

Type: **const UINT***

A pointer to a constant array of **UINT** containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

Requirements

Header	directml.h

See also

- [DML_MAX_POOLING1_OPERATOR_DESC](#)

DML_MAX_POOLING1_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a max pooling function across the input tensor (according to kernel sizes, stride sizes, and pad lengths), $y = \max(x_1 + x_2 + \dots + x_{\text{pool_size}})$.

Max pooling consists of computing the max on all values of a subset of the input tensor according to the kernel size, and then downsampling the data into the output tensor Y for further processing.

DML_MAX_POOLING1_OPERATOR_DESC is an updated version of [DML_MAX_POOLING_OPERATOR_DESC](#).

Syntax

```
struct DML_MAX_POOLING1_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    const DML_TENSOR_DESC *OutputIndicesTensor;  
    UINT DimensionCount;  
    const UINT *Strides;  
    const UINT *WindowSize;  
    const UINT *StartPadding;  
    const UINT *EndPadding;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from. This tensor's dimensions for the image case are [N, C, H, W], where N is the batch size, C is the number of channels, and H and W are the height and the width of the data. For the non-image case, the dimensions are in the form of [N, C, D1, D2, ..., Dn], where N is the batch size.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

`OutputIndicesTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant [DML_TENSOR_DESC](#) containing the output indices. The output indices state which maximal element from `InputTensor` was written to `OutputTensor`, where the indices are zero-based in terms of `InputTensor`, treated as if `InputTensor` were one large contiguous 1D array. Both `OutputTensor` and `OutputIndicesTensor` have the same size.

For example, given the input `[[3,5,7,1],[9,4,2,8]]` where the input indices are implicitly `[0,1,2,3,4,5,6,7]`, and with a pooling size of `3x1`, the output values would be `[[7,7],[9,8]]`, and the output indices would be `[[2,2],[4,7]]`. In `[[7,7],[4,7]]`, the `7` comes from index 2, the `9` comes from index 4, and the `8` comes from index 7.

`DimensionCount`

Type: `UINT`

The number of dimensions. This field determines the size of the `Strides`, `WindowSize`, `StartPadding`, and `EndPadding` arrays.

`Strides`

Type: `const UINT*`

A pointer to a constant array of `UINT` containing the lengths of the strides of the tensor.

`WindowSize`

Type: `const UINT*`

A pointer to a constant array of `UINT` containing the lengths of the pooling windows.

`StartPadding`

Type: `const UINT*`

A pointer to a constant array of `UINT` containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

`EndPadding`

Type: `const UINT*`

A pointer to a constant array of `UINT` containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	<code>directml.h</code>

See also

- [DML_MAX_POOLING_OPERATOR_DESC](#)

DML_MAX_UNPOOLING_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that fills the output tensor of the given shape (either explicit, or the input shape plus padding) with zeros, then writes each value from the input tensor into the output tensor at the element offset from the corresponding indices array.

`DML_MAX_UNPOOLING_OPERATOR_DESC` is the inverse of [DML_MAX_POOLING_OPERATOR_DESC](#).

All indices should be unique, and within bounds. The value written for duplicate indices is not defined; out-of-bounds indices yield no write.

Syntax

```
struct DML_MAX_UNPOOLING_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *IndicesTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the indices.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a mean variance normalization function on the input tensor.

Exponent = Const(2.0)

Epsilon = Const(1e-9)

X_RM = ReduceMean(X)

EX_squared = Pow(X_RM, Exponent)

X_squared = Pow(X, Exponent)

E_Xsquared = ReduceMean(X_squared)

Variance = Sub(E_Xsquared, EX_squared)

STD = Sqrt(Variance)

X_variance = Sub(X, X_RM)

Processed_STD = Add(STD, Epsilon)

X_MVN = Div(X_variance, Processed_STD)

Syntax

```
struct DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *ScaleTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *OutputTensor;
    BOOL                 CrossChannel;
    BOOL                 NormalizeVariance;
    FLOAT                Epsilon;
    const DML_OPERATOR_DESC *FusedActivation;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

ScaleTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the scale.

BiasTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the bias.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

CrossChannel

Type: **BOOL**

TRUE if the LRN layer is channel-wise (cross-channel). Otherwise, FALSE.

NormalizeVariance

Type: **BOOL**

TRUE if the variance should be normalized. Otherwise, FALSE.

Epsilon

Type: **FLOAT**

The epsilon value to use to avoid division by zero.

FusedActivation

Type: **const DML_OPERATOR_DESC***

An optional pointer to a constant **DML_OPERATOR_DESC** containing the fused activation layer.

Requirements

Header	
	directml.h

DML_ONE_HOT_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that generates a tensor with each element filled with two values—either an 'on' or an 'off' value. The 'on'/'off' values aren't limited to Boolean `true` and `false`, but whatever two values are contained in *ValuesTensor*.

The output tensor is one additional dimension larger than the input indices tensor, inserted along the desired axis. Indices out of bounds are ignored (leaving just 'off' values along that sliver).

Syntax

```
struct DML_ONE_HOT_OPERATOR_DESC {
    const DML_TENSOR_DESC *IndicesTensor;
    const DML_TENSOR_DESC *ValuesTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 Axis;
};
```

Members

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the indices.

`ValuesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the values.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

`Axis`

Type: `UINT`

The axis dimension to add to *OutputTensor*.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)

Header	directml.h
--------	------------

DML_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A generic container for an operator description. You construct DirectML operators using the parameters specified in this struct. See [IDMLDevice::CreateOperator](#) for additional details.

Syntax

```
struct DML_OPERATOR_DESC {  
    DML_OPERATOR_TYPE Type;  
    const void        *Desc;  
};
```

Members

Type

Type: [DML_OPERATOR_TYPE](#)

The type of the operator description. See [DML_OPERATOR_TYPE](#) for the available types.

Desc

Type: [const void*](#)

A pointer to the operator description. The type of the pointed-to struct must match the value specified in *Type*.

Requirements

Header	directml.h
--------	------------

DML_OPERATOR_TYPE enumeration

5/13/2020 • 5 minutes to read • [Edit Online](#)

Defines the type of an operator description.

See [DML_OPERATOR_DESC](#) for the usage of this enumeration.

Syntax

```
typedef enum DML_OPERATOR_TYPE {  
    DML_OPERATOR_INVALID,  
    DML_OPERATOR_ELEMENT_WISE_IDENTITY,  
    DML_OPERATOR_ELEMENT_WISE_ABS,  
    DML_OPERATOR_ELEMENT_WISE_ACOS,  
    DML_OPERATOR_ELEMENT_WISE_ADD,  
    DML_OPERATOR_ELEMENT_WISE_ASIN,  
    DML_OPERATOR_ELEMENT_WISE_ATAN,  
    DML_OPERATOR_ELEMENT_WISE_CEIL,  
    DML_OPERATOR_ELEMENT_WISE_CLIP,  
    DML_OPERATOR_ELEMENT_WISE_COS,  
    DML_OPERATOR_ELEMENT_WISE_DIVIDE,  
    DML_OPERATOR_ELEMENT_WISE_EXP,  
    DML_OPERATOR_ELEMENT_WISE_FLOOR,  
    DML_OPERATOR_ELEMENT_WISE_LOG,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_AND,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_NOT,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_OR,  
    DML_OPERATOR_ELEMENT_WISE_LOGICAL_XOR,  
    DML_OPERATOR_ELEMENT_WISE_MAX,  
    DML_OPERATOR_ELEMENT_WISE_MEAN,  
    DML_OPERATOR_ELEMENT_WISE_MIN,  
    DML_OPERATOR_ELEMENT_WISE_MULTIPLY,  
    DML_OPERATOR_ELEMENT_WISE_POW,  
    DML_OPERATOR_ELEMENT_WISE_CONSTANT_POW,  
    DML_OPERATOR_ELEMENT_WISE_RECIP,  
    DML_OPERATOR_ELEMENT_WISE_SIN,  
    DML_OPERATOR_ELEMENT_WISE_SQRT,  
    DML_OPERATOR_ELEMENT_WISE_SUBTRACT,  
    DML_OPERATOR_ELEMENT_WISE_TAN,  
    DML_OPERATOR_ELEMENT_WISE_THRESHOLD,  
    DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR,  
    DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR,  
    DML_OPERATOR_ACTIVATION_ELU,  
    DML_OPERATOR_ACTIVATION_HARDMAX,  
    DML_OPERATOR_ACTIVATION_HARD_SIGMOID,  
    DML_OPERATOR_ACTIVATION_IDENTITY,  
    DML_OPERATOR_ACTIVATION_LEAKY_RELU,  
    DML_OPERATOR_ACTIVATION_LINEAR,  
    DML_OPERATOR_ACTIVATION_LOG_SOFTMAX,  
    DML_OPERATOR_ACTIVATION_PARAMETERIZED_RELU,  
    DML_OPERATOR_ACTIVATION_PARAMETRIC_SOFTPLUS,  
    DML_OPERATOR_ACTIVATION_RELU,  
    DML_OPERATOR_ACTIVATION_SCALED_ELU,  
    DML_OPERATOR_ACTIVATION_SCALED_TANH,  
    DML_OPERATOR_ACTIVATION_SIGMOID,  
    DML_OPERATOR_ACTIVATION_SOFTMAX,  
    DML_OPERATOR_ACTIVATION_SOFTPLUS,  
    DML_OPERATOR_ACTIVATION_SOFTSIGN,
```

```

DML_OPERATOR_ACTIVATION_TANH,
DML_OPERATOR_ACTIVATION_THRESHOLDED_RELU,
DML_OPERATOR_CONVOLUTION,
DML_OPERATOR_GEMM,
DML_OPERATOR_REDUCE,
DML_OPERATOR_AVERAGE_POOLING,
DML_OPERATOR_LP_POOLING,
DML_OPERATOR_MAX_POOLING,
DML_OPERATOR_ROI_POOLING,
DML_OPERATOR_SLICE,
DML_OPERATOR_CAST,
DML_OPERATOR_SPLIT,
DML_OPERATOR_JOIN,
DML_OPERATOR_PADDING,
DML_OPERATOR_VALUE_SCALE_2D,
DML_OPERATOR_UPSAMPLE_2D,
DML_OPERATOR_GATHER,
DML_OPERATOR_SPACE_TO_DEPTH,
DML_OPERATOR_DEPTH_TO_SPACE,
DML_OPERATOR_TILE,
DML_OPERATOR_TOP_K,
DML_OPERATOR_BATCH_NORMALIZATION,
DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION,
DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION,
DML_OPERATOR_LP_NORMALIZATION,
DML_OPERATOR_RNN,
DML_OPERATOR_LSTM,
DML_OPERATOR_GRU,
DML_OPERATOR_ELEMENT_WISE_SIGN,
DML_OPERATOR_ELEMENT_WISE_IS_NAN,
DML_OPERATOR_ELEMENT_WISE_ERF,
DML_OPERATOR_ELEMENT_WISE_SINH,
DML_OPERATOR_ELEMENT_WISE_COSH,
DML_OPERATOR_ELEMENT_WISE_TANH,
DML_OPERATOR_ELEMENT_WISE_ASINH,
DML_OPERATOR_ELEMENT_WISE_ACOSH,
DML_OPERATOR_ELEMENT_WISE_ATANH,
DML_OPERATOR_ELEMENT_WISE_IF,
DML_OPERATOR_ELEMENT_WISE_ADD1,
DML_OPERATOR_ACTIVATION_SHRINK,
DML_OPERATOR_MAX_POOLING1,
DML_OPERATOR_MAX_UNPOOLING,
DML_OPERATOR_DIAGONAL_MATRIX,
DML_OPERATOR_SCATTER,
DML_OPERATOR_ONE_HOT,
DML_OPERATOR_RESAMPLE
} ;

```

Constants

DML_OPERATOR_INVALID	Indicates an unknown operator type, and is never valid. Using this value results in an error.
DML_OPERATOR_ELEMENT_WISE_IDENTITY	Indicates the operator described by the DML_ELEMENT_WISE_IDENTITY_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ABS	Indicates the operator described by the DML_ELEMENT_WISE_ABS_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ACOS	Indicates the operator described by the DML_ELEMENT_WISE_ACOS_OPERATOR_DESC structure.

DML_OPERATOR_ELEMENT_WISE_ADD	Indicates the operator described by the DML_ELEMENT_WISE_ADD_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ASIN	Indicates the operator described by the DML_ELEMENT_WISE_ASIN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ATAN	Indicates the operator described by the DML_ELEMENT_WISE_ATAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_CEIL	Indicates the operator described by the DML_ELEMENT_WISE_CEIL_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_CLIP	Indicates the operator described by the DML_ELEMENT_WISE_CLIP_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_COS	Indicates the operator described by the DML_ELEMENT_WISE_COS_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_DIVIDE	Indicates the operator described by the DML_ELEMENT_WISE_DIVIDE_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_EXP	Indicates the operator described by the DML_ELEMENT_WISE_EXP_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_FLOOR	Indicates the operator described by the DML_ELEMENT_WISE_FLOOR_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOG	Indicates the operator described by the DML_ELEMENT_WISE_LOG_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_AND	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_AND_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_EQUALS	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_EQUALS_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_GREATER_THAN	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_GREATER_THAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_LESS_THAN	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_LESS_THAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_NOT	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_NOT_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_LOGICAL_OR	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_OR_OPERATOR_DESC structure.

DML_OPERATOR_ELEMENT_WISE_LOGICAL_XOR	Indicates the operator described by the DML_ELEMENT_WISE_LOGICAL_XOR_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MAX	Indicates the operator described by the DML_ELEMENT_WISE_MAX_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MEAN	Indicates the operator described by the DML_ELEMENT_WISE_MEAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MIN	Indicates the operator described by the DML_ELEMENT_WISE_MIN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_MULTIPLY	Indicates the operator described by the DML_ELEMENT_WISE_MULTIPLY_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_POW	Indicates the operator described by the DML_ELEMENT_WISE_POW_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_CONSTANT_POW	Indicates the operator described by the DML_ELEMENT_WISE_CONSTANT_POW_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_RECIP	Indicates the operator described by the DML_ELEMENT_WISE_RECIP_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_SIN	Indicates the operator described by the DML_ELEMENT_WISE_SIN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_SQRT	Indicates the operator described by the DML_ELEMENT_WISE_SQRT_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_SUBTRACT	Indicates the operator described by the DML_ELEMENT_WISE_SUBTRACT_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_TAN	Indicates the operator described by the DML_ELEMENT_WISE_TAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_THRESHOLD	Indicates the operator described by the DML_ELEMENT_WISE_THRESHOLD_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_QUANTIZE_LINEAR	Indicates the operator described by the DML_ELEMENT_WISE_QUANTIZE_LINEAR_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_DEQUANTIZE_LINEAR	Indicates the operator described by the DML_ELEMENT_WISE_DEQUANTIZE_LINEAR_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_ELU	Indicates the operator described by the DML_ACTIVATION_ELU_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_HARDMAX	Indicates the operator described by the DML_ACTIVATION_HARDMAX_OPERATOR_DESC structure.

DML_OPERATOR_ACTIVATION_HARD_SIGMOID	Indicates the operator described by the DML_ACTIVATION_HARD_SIGMOID_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_IDENTITY	Indicates the operator described by the DML_ACTIVATION_IDENTITY_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_LEAKY_RELU	Indicates the operator described by the DML_ACTIVATION_LEAKY_RELU_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_LINEAR	Indicates the operator described by the DML_ACTIVATION_LINEAR_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_LOG_SOFTMAX	Indicates the operator described by the DML_ACTIVATION_LOG_SOFTMAX_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_PARAMETERIZED_RELU	Indicates the operator described by the DML_ACTIVATION_PARAMETERIZED_RELU_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_PARAMETRIC_SOFTPLUS	Indicates the operator described by the DML_ACTIVATION_PARAMETRIC_SOFTPLUS_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_RELU	Indicates the operator described by the DML_ACTIVATION_RELU_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SCALED_ELU	Indicates the operator described by the DML_ACTIVATION_SCALED_ELU_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SCALED_TANH	Indicates the operator described by the DML_ACTIVATION_SCALED_TANH_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SIGMOID	Indicates the operator described by the DML_ACTIVATION_SIGMOID_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SOFTMAX	Indicates the operator described by the DML_ACTIVATION_SOFTMAX_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SOFTPLUS	Indicates the operator described by the DML_ACTIVATION_SOFTPLUS_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SOFTSIGN	Indicates the operator described by the DML_ACTIVATION_SOFTSIGN_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_TANH	Indicates the operator described by the DML_ACTIVATION_TANH_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_THRESHOLDED_RELU	Indicates the operator described by the DML_ACTIVATION_THRESHOLDED_RELU_OPERATOR_DESC structure.
DML_OPERATOR_CONVOLUTION	Indicates the operator described by the DML_CONVOLUTION_OPERATOR_DESC structure.

DML_OPERATOR_GEMM	Indicates the operator described by the DML_GEMM_OPERATOR_DESC structure.
DML_OPERATOR_REDUCE	Indicates the operator described by the DML_REDUCE_OPERATOR_DESC structure.
DML_OPERATOR_AVERAGE_POOLING	Indicates the operator described by the DML_AVERAGE_POOLING_OPERATOR_DESC structure.
DML_OPERATOR_LP_POOLING	Indicates the operator described by the DML_LP_POOLING_OPERATOR_DESC structure.
DML_OPERATOR_MAX_POOLING	Indicates the operator described by the DML_MAX_POOLING_OPERATOR_DESC structure.
DML_OPERATOR_ROI_POOLING	Indicates the operator described by the DML_ROI_POOLING_OPERATOR_DESC structure.
DML_OPERATOR_SLICE	Indicates the operator described by the DML_SLICE_OPERATOR_DESC structure.
DML_OPERATOR_CAST	Indicates the operator described by the DML_CAST_OPERATOR_DESC structure.
DML_OPERATOR_SPLIT	Indicates the operator described by the DML_SPLIT_OPERATOR_DESC structure.
DML_OPERATOR_JOIN	Indicates the operator described by the DML_JOIN_OPERATOR_DESC structure.
DML_OPERATOR_PADDING	Indicates the operator described by the DML_PADDING_OPERATOR_DESC structure.
DML_OPERATOR_VALUE_SCALE_2D	Indicates the operator described by the DML_VALUE_SCALE_2D_OPERATOR_DESC structure.
DML_OPERATOR_UPSAMPLE_2D	Indicates the operator described by the DML_UPSAMPLE_2D_OPERATOR_DESC structure.
DML_OPERATOR_GATHER	Indicates the operator described by the DML_GATHER_OPERATOR_DESC structure.
DML_OPERATOR_SPACE_TO_DEPTH	Indicates the operator described by the DML_SPACE_TO_DEPTH_OPERATOR_DESC structure.
DML_OPERATOR_DEPTH_TO_SPACE	Indicates the operator described by the DML_DEPTH_TO_SPACE_OPERATOR_DESC structure.
DML_OPERATOR_TILE	Indicates the operator described by the DML_TILE_OPERATOR_DESC structure.
DML_OPERATOR_TOP_K	Indicates the operator described by the DML_TOP_K_OPERATOR_DESC structure.

DML_OPERATOR_BATCH_NORMALIZATION	Indicates the operator described by the DML_BATCH_NORMALIZATION_OPERATOR_DESC structure.
DML_OPERATOR_MEAN_VARIANCE_NORMALIZATION	Indicates the operator described by the DML_MEAN_VARIANCE_NORMALIZATION_OPERATOR_DESC structure.
DML_OPERATOR_LOCAL_RESPONSE_NORMALIZATION	Indicates the operator described by the DML_LOCAL_RESPONSE_NORMALIZATION_OPERATOR_DESC structure.
DML_OPERATOR_LP_NORMALIZATION	Indicates the operator described by the DML_LP_NORMALIZATION_OPERATOR_DESC structure.
DML_OPERATOR_RNN	Indicates the operator described by the DML_RNN_OPERATOR_DESC structure.
DML_OPERATOR_LSTM	Indicates the operator described by the DML_LSTM_OPERATOR_DESC structure.
DML_OPERATOR_GRU	Indicates the operator described by the DML_GRU_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_SIGN	Indicates the operator described by the DML_ELEMENT_WISE_SIGN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_IS_NAN	Indicates the operator described by the DML_ELEMENT_WISE_IS_NAN_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ERF	Indicates the operator described by the DML_ELEMENT_WISE_ERF_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_SINH	Indicates the operator described by the DML_ELEMENT_WISE_SINH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_COSH	Indicates the operator described by the DML_ELEMENT_WISE_COSH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_TANH	Indicates the operator described by the DML_ELEMENT_WISE_TANH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ASINH	Indicates the operator described by the DML_ELEMENT_WISE_ASINH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ACOSH	Indicates the operator described by the DML_ELEMENT_WISE_ACOSH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_ATANH	Indicates the operator described by the DML_ELEMENT_WISE_ATANH_OPERATOR_DESC structure.
DML_OPERATOR_ELEMENT_WISE_IF	Indicates the operator described by the DML_ELEMENT_WISE_IF_OPERATOR_DESC structure.

DML_OPERATOR_ELEMENT_WISE_ADD1	Indicates the operator described by the DML_ELEMENT_WISE_ADD1_OPERATOR_DESC structure.
DML_OPERATOR_ACTIVATION_SHRINK	Indicates the operator described by the DML_ACTIVATION_SHRINK_OPERATOR_DESC structure.
DML_OPERATOR_MAX_POOLING1	Indicates the operator described by the DML_MAX_POOLING1_OPERATOR_DESC structure.
DML_OPERATOR_MAX_UNPOOLING	Indicates the operator described by the DML_MAX_UNPOOLING_OPERATOR_DESC structure.
DML_OPERATOR_DIAGONAL_MATRIX	Indicates the operator described by the DML_DIAGONAL_MATRIX_OPERATOR_DESC structure.
DML_OPERATOR_SCATTER	Indicates the operator described by the DML_SCATTER_OPERATOR_DESC structure.
DML_OPERATOR_ONE_HOT	Indicates the operator described by the DML_ONE_HOT_OPERATOR_DESC structure.
DML_OPERATOR_RESAMPLE	Indicates the operator described by the DML_RESAMPLE_OPERATOR_DESC structure.

Requirements

Header	directml.h
--------	------------

DML_PADDING_MODE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a mode for the DirectML pad operator (as described by the [DML_PADDING_OPERATOR_DESC](#) structure).

Syntax

```
typedef enum DML_PADDING_MODE {  
    DML_PADDING_MODE_CONSTANT,  
    DML_PADDING_MODE_EDGE,  
    DML_PADDING_MODE_REFLECTION  
} ;
```

Constants

DML_PADDING_MODE_CONSTANT	Indicates padding with a constant.
DML_PADDING_MODE_EDGE	Indicates edge mode for padding.
DML_PADDING_MODE_REFLECTION	Indicates reflection mode for padding.

Requirements

Header	directml.h
--------	------------

See also

[DML_PADDING_OPERATOR_DESC](#)

DML_PADDING_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that inflates the input tensor with zeroes (or some other value) on the edges.

Syntax

```
struct DML_PADDING_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_PADDING_MODE     PaddingMode;
    FLOAT                PaddingValue;
    UINT                 DimensionCount;
    const UINT            *StartPadding;
    const UINT            *EndPadding;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

PaddingMode

Type: **DML_PADDING_MODE**

The padding mode to use.

PaddingValue

Type: **FLOAT**

The value with which to pad.

DimensionCount

Type: **UINT**

The number of dimensions. This field determines the size of the *StartPadding* and *EndPadding* arrays.

StartPadding

Type: **const UINT***

A pointer to a constant array of **UINT** containing the padding (number of pixels added) to the start of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

EndPadding

Type: **const [UINT](#)***

A pointer to a constant array of [UINT](#) containing the padding (number of pixels added) to the end of the corresponding axis. Padding defaults to 0 along the start and end of each axis.

Requirements

Header	
	directml.h

DML_RECURRENT_NETWORK_DIRECTION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify a direction for a recurrent DirectML operator.

Syntax

```
typedef enum DML_RECURRENT_NETWORK_DIRECTION {  
    DML_RECURRENT_NETWORK_DIRECTION_FORWARD,  
    DML_RECURRENT_NETWORK_DIRECTION_BACKWARD,  
    DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL  
} ;
```

Constants

DML_RECURRENT_NETWORK_DIRECTION_FORWARD	Indicates the forward pass.
DML_RECURRENT_NETWORK_DIRECTION_BACKWARD	Indicates the backward pass.
DML_RECURRENT_NETWORK_DIRECTION_BIDIRECTIONAL	Indicates both passes.

Requirements

Header	directml.h

DML_REDUCE_FUNCTION enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Defines constants that specify the specific reduction algorithm to use for the DirectML reduce operator (as described by the [DML_REDUCE_OPERATOR_DESC](#) structure).

Syntax

```
typedef enum DML_REDUCE_FUNCTION {
    DML_REDUCE_FUNCTION_ARGMAX,
    DML_REDUCE_FUNCTION_ARGMIN,
    DML_REDUCE_FUNCTION_AVERAGE,
    DML_REDUCE_FUNCTION_L1,
    DML_REDUCE_FUNCTION_L2,
    DML_REDUCE_FUNCTION_LOG_SUM,
    DML_REDUCE_FUNCTION_LOG_SUM_EXP,
    DML_REDUCE_FUNCTION_MAX,
    DML_REDUCE_FUNCTION_MIN,
    DML_REDUCE_FUNCTION_MULTIPLY,
    DML_REDUCE_FUNCTION_SUM,
    DML_REDUCE_FUNCTION_SUM_SQUARE
} ;
```

Constants

DML_REDUCE_FUNCTION_ARGMAX	Indicates a reduction function that computes the indices of the max elements of the input tensor's elements along the specified axis, int32 {i j k ..} = maxindex(X Y Z ...).						

DML_REDUCE_FUNCTION_ARGMIN	Indicates a reduction function that computes the indices of the min elements of the input tensor's elements along the specified axis, int32 {i j k ..} = minindex(X Y Z ...).						
DML_REDUCE_FUNCTION_AVERAGE	Indicates a reduction function that computes the mean of the input tensor's elements along the specified axes, x = (x1 + x2 + ... + xn) / n.						
DML_REDUCE_FUNCTION_L1	Indicates a reduction function that computes the L1 norm of the input tensor's elements along the specified axes, x =	x1	+	x2	+ ... +	xn	.

DML_REDUCE_FUNCTION_L2	Indicates a reduction function that computes the L2 norm of the input tensor's elements along the specified axes, $x = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$.				
DML_REDUCE_FUNCTION_LOG_SUM	Indicates a reduction function that computes the log sum of the input tensor's elements along the specified axes, $x = \log(x_1 + x_2 + \dots + x_n)$.				
DML_REDUCE_FUNCTION_LOG_SUM_EXP	Indicates a reduction function that computes the log sum exponent of the input tensor's elements along the specified axes, $x = \log(\exp(x_1) + \exp(x_2) + \dots + \exp(x_n))$.				

DML_REDUCE_FUNCTION_MAX	Indicates a reduction function that computes the max of the input tensor's elements along the specified axes, $x = \max(\max(m \text{ ax}(x_1, x_2), x_3), \dots, x_n)$.				
DML_REDUCE_FUNCTION_MIN	Indicates a reduction function that computes the min of the input tensor's elements along the specified axes, $x = \min(\min(m \text{ ni}(x_1, x_2), x_3), \dots, x_n)$.				
DML_REDUCE_FUNCTION_MULTIPLY	Indicates a reduction function that computes the product of the input tensor's elements along the specified axes, $x = (x_1 * x_2 * \dots * x_n)$.				
DML_REDUCE_FUNCTION_SUM	Indicates a reduction function that computes the sum of the input tensor's elements along the specified axes, $x = (x_1 + x_2 + \dots + x_n)$.				

DML_REDUCE_FUNCTION_SUM_SQUARE	Indicates a reduction function that computes the sum square of the input tensor's elements along the specified axes, $x = x_1^2 + x_2^2 + \dots + x_n^2$.				
--------------------------------	--	--	--	--	--

Requirements

Header	directml.h
--------	------------

See also

[DML_REDUCE_OPERATOR_DESC](#)

DML_REDUCE_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs the specified reduction function on the input.

Syntax

```
struct DML_REDUCE_OPERATOR_DESC {  
    DML_REDUCE_FUNCTION    Function;  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT                  AxisCount;  
    const UINT             *Axes;  
};
```

Members

Function

Type: [DML_REDUCE_FUNCTION](#)

A [DML_REDUCE_FUNCTION](#) value specifying the reduce function to apply to the input.

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from.

OutputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to.

AxisCount

Type: [UINT](#)

The number of axes. This field determines the size of the *Axes* array.

Axes

Type: [const UINT*](#)

A pointer to a constant array of [UINT](#) containing the axes along which to reduce.

Requirements

Header

[directml.h](#)

DML_RESAMPLE_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that resamples elements from the source to the destination tensor, using the scale factors to compute the destination tensor size. You can use a linear or nearest neighbor interpolation mode. The operator supports interpolation across multiple dimensions, not just 2D. So you can keep the same spatial size, but interpolate across channels or across batches.

Syntax

```
struct DML_RESAMPLE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_INTERPOLATION_MODE InterpolationMode;
    UINT ScaleCount;
    const FLOAT *Scales;
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

`InterpolationMode`

Type: `DML_INTERPOLATION_MODE`

The interpolation mode to use for the upsample. You can use the default value of `DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR`.

`ScaleCount`

Type: `UINT`

The size of the `Scales` array.

`Scales`

Type: `const FLOAT*`

A pointer to a constant array of `FLOAT` containing the scale factors.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_RNN_OPERATOR_DESC structure

5/13/2020 • 4 minutes to read • [Edit Online](#)

Describes a DirectML deep learning operator that performs a one-layer simple recurrent neural network (RNN) function on the input. This function is often referred to as the *Input Gate*. This operator performs this function multiple times in a loop, dictated by the sequence length dimension and the *SequenceLengthsTensor* argument.

Equation for the forward direction

```
for (t = 0; t < seq_length; t++)
{
    $H_t = f(X_t * W_i^T + H_{t-1} * R_i^T + W_{bi} + R_{bi})$
```

Equation for the backward direction

```
for (t = seq_length - 1; t >= 0; t--)
{
    $H_t = f(X_t * W_{Bi}^T + H_{t-1} * R_{Bi}^T + W_{Bbi} + R_{Bbi})$
```

Equation legend

- \$t\$ current RNN layer index.
- \$t-1\$ previous RNN layer index. If backwards direction, then it means \$t+1\$ index but still the previously calculated \$t\$.
- \$T\$ signifies that a matrix will be transposed before use.
- \$*\$ signifies a matrix multiplication.
- \$+\$ signifies a element-wise addition of two matrices.
- \$H_t\$ output of current RNN index \$t\$.
- \$f()\$ activation function dictated by *ActivationDescs*.
- \$X_t\$ Sub-matrix of the *InputTensor*, which is defined by matrix index [\$t\$, batch_size, input_size].
- \$W_i^T\$ Forward sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- \$W_{Bi}^T\$ Backwards sub-matrix defined in *WeightTensor*, which is defined to be size [1, batch_size, input_size]. This matrix will be transposed before use.
- \$H_{t-1}\$ an intermediate tensor that's defined to be the output of the previous layer. For \$t = 0\$, \$H_{t-1}\$ is replaced with *HiddenInitTensor*. This is defined to be size [1, batch_size, hidden_size]. A different *HiddenInitTensor* sub-matrix is used for each direction.
- \$R_i^T\$ Forward sub-matrix of *RecurrenceTensor*, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- \$R_{Bi}^T\$ Backwards sub-matrix of *RecurrenceTensor*, which is defined to be size [1, hidden_size, hidden_size]. This matrix will be transposed before use.
- \$W_{bi}\$, \$R_{bi}\$ are the bias tensors of the weight tensor, and recurrence tensor, which are one of the sub-matrices of *BiasTensor*. This matrix is size [1,hidden_size] and is broadcasted up to the required size which is [1, batch_size, hidden_size].
- \$W_{Bbi}\$, \$R_{Bbi}\$ are the bias tensors for the backwards direction
- \$i\$ stands for input gate.

Syntax

```
struct DML_RNN_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *WeightTensor;
    const DML_TENSOR_DESC *RecurrenceTensor;
    const DML_TENSOR_DESC *BiasTensor;
    const DML_TENSOR_DESC *HiddenInitTensor;
    const DML_TENSOR_DESC *SequenceLengthsTensor;
    const DML_TENSOR_DESC *OutputSequenceTensor;
    const DML_TENSOR_DESC *OutputSingleTensor;
    UINT ActivationDescCount;
    const DML_OPERATOR_DESC *ActivationDescs;
    DML_RECURRENT_NETWORK_DIRECTION Direction;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from, \$X\$. Packed (and potentially padded) into one 4-D tensor with the shape of [1, seq_length, batch_size, input_size]. seq_length is the dimension that is mapped to the RNN index, \$t\$.

WeightTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the weight tensor for the gates, \$W\$. Concatenation of \$W_i\$ and \$W_{Bi}\$ (if bidirectional). The tensor has shape [1, num_directions, hidden_size, input_size].

RecurrenceTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the recurrence weight tensor, \$R\$. Concatenation of \$R_i\$ and \$R_{Bi}\$ (if bidirectional). This tensor has shape [1, num_directions, hidden_size, hidden_size].

BiasTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the bias tensor for the input gate, \$B\$. Concatenation of [\$W_{Bi}\$, \$R_{Bi}\$], and [\$W_{Bbi}\$, \$R_{Bbi}\$] (if bidirectional). This tensor has shape [1, 1, num_directions, 2 * hidden_size]. If not specified, then defaults to 0.

HiddenInitTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing the hidden node initializer tensor, \$H_{t-1}\$ for the first loop index \$t\$. If not specified, then defaults to 0. This tensor has shape [1, num_directions, batch_size, hidden_size].

SequenceLengthsTensor

Type: **const DML_TENSOR_DESC***

An optional pointer to a constant **DML_TENSOR_DESC** containing an independent seq_length for each element in

the batch. If not specified, then all sequences in the batch have length seq_length. This tensor has shape [1, 1, 1, batch_size].

`OutputSequenceTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the concatenation of all the intermediate layer output values of the hidden nodes, H_t . This tensor has shape [seq_length, num_directions, batch_size, hidden_size]. seq_length is mapped to the loop index t .

`OutputSingleTensor`

Type: `const DML_TENSOR_DESC*`

An optional pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to which to write the final output value of the hidden nodes, H_t . This tensor has shape [1, num_directions, batch_size, hidden_size].

`ActivationDescCount`

Type: `UINT`

This field determines the size of the *ActivationDescs* array.

`ActivationDescs`

Type: `const DML_OPERATOR_DESC*`

A pointer to a constant array of `DML_OPERATOR_DESC` containing the descriptions of the activation operators, $f()$. The number of activation functions is equal to the number of directions. For forwards and backwards directions there is expected to be 1 activation function. For Bidirectional there are expected to be 2.

`Direction`

Type: `const DML_RECURRENT_NETWORK_DIRECTION*`

The direction of the operator—forward, backwards, or bidirectional. You can use a default value of `DML_RECURRENT_NETWORK_DIRECTION_FORWARD`.

Requirements

Header	directml.h

DML_ROI_POOLING_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs a pooling function across the input tensor (according to regions of interest, or ROIs).

Syntax

```
struct DML_ROI_POOLING_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *ROITensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT SpatialScale;  
    DML_SIZE_2D PooledSize;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from. This tensor's dimensions are [N, C, H, W], where N is the batch size, C is the number of channels, and H and W are the height and the width of the data.

ROITensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the regions of interest (ROIs) to pool over. This tensor is a 2-D tensor of shape (num_rois, 5) given as [[batch_id, x1, y1, x2, y2], ...].

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to. This tensor is the ROI-pooled output; a 4-D tensor of shape (num_rois, channels, pooled_size[0], pooled_size[1]).

SpatialScale

Type: **FLOAT**

Multiplicative spatial scale factor, used to translate the ROI coordinates from their input scale to the scale used when pooling. You can use a default value of 1.0.

PooledSize

Type: **DML_SIZE_2D**

The ROI pool output size (height, width).

Requirements

Header	directml.h
---------------	------------

DML_SCALE_BIAS structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Contains the values of scale and bias terms supplied to a DirectML operator. Scale and bias have the effect of applying the function $g(x) = x * \text{Scale} + \text{Bias}$.

Syntax

```
struct DML_SCALE_BIAS {  
    FLOAT Scale;  
    FLOAT Bias;  
};
```

Members

Scale

Type: [FLOAT](#)

The scale term in $g(x) = x * \text{Scale} + \text{Bias}$.

Bias

Type: [FLOAT](#)

The bias term in $g(x) = x * \text{Scale} + \text{Bias}$.

Requirements

Header	directml.h
--------	------------

DML_SCATTER_OPERATOR_DESC structure

5/13/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that copies the whole input tensor to the output, then overwrites selected indices with corresponding values from the updates tensor.

`DML_SCATTER_OPERATOR_DESC` is the inverse of [DML_GATHER_OPERATOR_DESC](#).

If two output element indices overlap (which is invalid), then only the last write takes effect.

Syntax

```
struct DML_SCATTER_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *IndicesTensor;  
    const DML_TENSOR_DESC *UpdatesTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT                 Axis;  
};
```

Members

`InputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to read from.

`IndicesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the indices.

`UpdatesTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the updates.

`OutputTensor`

Type: `const DML_TENSOR_DESC*`

A pointer to a constant `DML_TENSOR_DESC` containing the description of the tensor to write the results to.

`Axis`

Type: `UINT`

The axis dimension to add to `OutputTensor`.

Requirements

Minimum supported client	Windows 10, version 2004 (10.0; Build 19041)
Minimum supported server	Windows Server, version 2004 (10.0; Build 19041)
Header	directml.h

DML_SIZE_2D structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Contains values that can represent the size (as supplied to a DirectML operator) of a 2-D plane of elements within a tensor, or a 2-D scale, or any 2-D width/height value.

Syntax

```
struct DML_SIZE_2D {  
    UINT Width;  
    UINT Height;  
};
```

Members

Width

Type: [UINT](#)

The width.

Height

Type: [UINT](#)

The height.

Requirements

Header	directml.h
--------	------------

DML_SLICE_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that extracts a single subregion (a "slice") of an input tensor.

This operator copies elements from the input tensor in each dimension N starting from the Offset[N], and advancing Strides[N] elements on the input tensor until Sizes[N] elements are copied to the output tensor.

For example a slice with Offsets [0 0 0 10], Sizes [1 1 1 5], and Strides [1 1 1 2] will copy every second element starting at the 10th element of the input tensor, until a total of 5 elements are copied to the output tensor.

Syntax

```
struct DML_SLICE_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT DimensionCount;  
    const UINT *Offsets;  
    const UINT *Sizes;  
    const UINT *Strides;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to extract slices from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the sliced data results to.

DimensionCount

Type: **UINT**

The number of dimensions. This field determines the size of the *Offsets*, *Sizes*, and *Strides* arrays. This value must match the DimensionCount of the input and output tensors.

Offsets

Type: **const UINT***

A pointer to a constant array of **UINT** containing the starting index of each dimension to slice from the input tensor.

Sizes

Type: **const UINT***

A pointer to a constant array of **UINT** containing the size of each dimension of the slice, in elements. The values in this array must match the sizes specified in the output tensor.

Strides

Type: **const [UINT](#)***

A pointer to a constant array of [UINT](#) containing the element strides for each dimension in the slice.

Requirements

Header	directml.h

DML_SPACE_TO_DEPTH_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that rearranges blocks of spatial data into depth. The operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimension.

This is the reverse transformation of [DML_DEPTH_TO_SPACE_OPERATOR_DESC](#).

Syntax

```
struct DML_SPACE_TO_DEPTH_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    UINT BlockSize;  
};
```

Members

InputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to read from. The input tensor's dimensions are [N,C,H,W], where N is the batch axis, C is the channel or depth, H is the height, and W is the width.

OutputTensor

Type: [const DML_TENSOR_DESC*](#)

A pointer to a constant [DML_TENSOR_DESC](#) containing the description of the tensor to write the results to. The output tensor's dimensions are [N, C * (BlockSize * BlockSize), H / BlockSize, W / BlockSize].

BlockSize

Type: [UINT](#)

Blocks of [BlockSize, BlockSize] are moved.

Requirements

Header	directml.h

See also

[DML_DEPTH_TO_SPACE_OPERATOR_DESC](#)

DML_SPLIT_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that splits the input tensor into multiple output tensors, along the specified axis.

Syntax

```
struct DML_SPLIT_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    UINT OutputCount;
    const DML_TENSOR_DESC *OutputTensors;
    UINT Axis;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputCount

Type: **UINT**

This field determines the size of the *OutputTensors* array.

OutputTensors

Type: **const DML_TENSOR_DESC***

A pointer to a constant array of **DML_TENSOR_DESC** containing the descriptions of the tensors to write the results to.

Axis

Type: **UINT**

The axis to split on.

Requirements

Header

directml.h

DML_TENSOR_DATA_TYPE enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies the data type of the values in a tensor. DirectML operators may not support all data types; see the documentation for each specific operator to find which data types it supports.

Syntax

```
typedef enum DML_TENSOR_DATA_TYPE {  
    DML_TENSOR_DATA_TYPE_UNKNOWN,  
    DML_TENSOR_DATA_TYPE_FLOAT32,  
    DML_TENSOR_DATA_TYPE_FLOAT16,  
    DML_TENSOR_DATA_TYPE_UINT32,  
    DML_TENSOR_DATA_TYPE_UINT16,  
    DML_TENSOR_DATA_TYPE_UINT8,  
    DML_TENSOR_DATA_TYPE_INT32,  
    DML_TENSOR_DATA_TYPE_INT16,  
    DML_TENSOR_DATA_TYPE_INT8  
} ;
```

Constants

DML_TENSOR_DATA_TYPE_UNKNOWN	Indicates an unknown data type. This value is never valid.
DML_TENSOR_DATA_TYPE_FLOAT32	Indicates a 32-bit floating-point data type.
DML_TENSOR_DATA_TYPE_FLOAT16	Indicates a 16-bit floating-point data type.
DML_TENSOR_DATA_TYPE_UINT32	Indicates a 32-bit unsigned integer data type.
DML_TENSOR_DATA_TYPE_UINT16	Indicates a 16-bit unsigned integer data type.
DML_TENSOR_DATA_TYPE_UINT8	Indicates a 8-bit unsigned integer data type.
DML_TENSOR_DATA_TYPE_INT32	Indicates a 32-bit signed integer data type.
DML_TENSOR_DATA_TYPE_INT16	Indicates a 16-bit signed integer data type.
DML_TENSOR_DATA_TYPE_INT8	Indicates a 8-bit signed integer data type.

Requirements

Header	directml.h
--------	------------

DML_TENSOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

A generic container for a DirectML tensor description.

Syntax

```
struct DML_TENSOR_DESC {  
    DML_TENSOR_TYPE Type;  
    const void      *Desc;  
};
```

Members

Type

Type: [DML_TENSOR_TYPE](#)

The type of the tensor description. See [DML_TENSOR_TYPE](#) for the available types.

Desc

Type: `const void*`

A pointer to the tensor description. The type of the pointed-to struct must match the value specified in *Type*.

Requirements

Header

`directml.h`

DML_TENSOR_FLAGS enumeration

5/27/2020 • 2 minutes to read • [Edit Online](#)

Specifies additional options in a tensor description. Values can be bitwise OR'd together.

Syntax

```
typedef enum DML_TENSOR_FLAGS {  
    DML_TENSOR_FLAG_NONE,  
    DML_TENSOR_FLAG_OWNED_BY_DML  
} ;
```

Constants

DML_TENSOR_FLAG_NONE	No options are specified.
DML_TENSOR_FLAG_OWNED_BY_DML	<p>Indicates that the tensor data should be owned and managed by DirectML. The effect of this flag is that DirectML makes a copy of the tensor data during initialization of an operator, storing it in the persistent resource. This allows DirectML to perform reformatting of the tensor data into other, more efficient forms. Setting this flag may increase performance, but is typically only useful for tensors whose data doesn't change for the lifetime of the operator (for example, weight tensors).</p> <p>This flag can only be used on input tensors.</p> <p>When this flag is set on a particular tensor description, the corresponding tensor must be bound to the binding table during operator initialization, and not during execution. Attempting to bind the tensor during execution while this flag is set results in an error. This is the opposite of the default behavior (the behavior without the DML_TENSOR_FLAG_OWNED_BY_DML flag), where the tensor is expected to be bound during execution, and not during initialization.</p>

Requirements

Header	directml.h
--------	------------

See also

[Binding in DirectML](#)

DML_TENSOR_TYPE enumeration

5/13/2020 • 2 minutes to read • [Edit Online](#)

Identifies a type of tensor description.

Syntax

```
typedef enum DML_TENSOR_TYPE {  
    DML_TENSOR_TYPE_INVALID,  
    DML_TENSOR_TYPE_BUFFER  
} ;
```

Constants

DML_TENSOR_TYPE_INVALID	Indicates an unknown tensor description type. This value is never valid.
DML_TENSOR_TYPE_BUFFER	Indicates a tensor description that is represented by a Direct3D 12 buffer. The corresponding struct type is DML_BUFFER_TENSOR_DESC .

Requirements

Header	directml.h
--------	------------

DML_TILE_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML data reorganization operator that constructs an output tensor by tiling the input tensor.

For example, InputTensor = [[1, 2], [3, 4]], Repeats = [1, 2], gives OutputTensor = [[1, 2, 1, 2], [3, 4, 3, 4]].

Syntax

```
struct DML_TILE_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    UINT                 RepeatsCount;
    const UINT            *Repeats;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from; this can be of any shape.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to. This tensor is of the same dimension and type as the input tensor. $\text{Output_dim}[i] = \text{input_dim}[i] * \text{repeats}[i]$.

RepeatsCount

Type: **UINT**

This field determines the size of the *Repeats* array.

Repeats

Type: **const UINT***

Each value in this array corresponds to one of the input tensor's dimensions (in order). Each value is the number of tiled copies to make of that dimension.

Requirements

Header

directml.h

DML_TOP_K_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML reduction operator that retrieves the top K elements along a specified axis.

Given an input tensor of shape [a_1, a_2, ..., a_n, r] and integer argument k, the operator returns two outputs, as detailed in the descriptions of the output parameters below. Given two equivalent values, the operator uses the indices along the axis as a tiebreaker. That is, the element with the lower index appears first.

Syntax

```
struct DML_TOP_K_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputValueTensor;  
    const DML_TENSOR_DESC *OutputIndexTensor;  
    UINT                 Axis;  
    UINT                 K;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputValueTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the resulting values to. This is a tensor of shape [a_1, a_2, ..., a_{axis-1}, k, a_{axis+1}, ... a_n], containing the top K values from the input tensor along the specified axis.

OutputIndexTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the resulting indices to. This is a tensor of shape [a_1, a_2, ..., a_{axis-1}, k, a_{axis+1}, ... a_n], containing the corresponding input tensor indices for the top K values.

Axis

Type: **UINT**

The dimension on which to do the sort.

K

Type: **UINT**

The number of top elements to retrieve.

Requirements

Header	directml.h

DML_UPSAMPLE_2D_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML imaging operator that upsamples the image contained in the input tensor. Each dimension value of the output tensor is $\text{output_dimension} = \text{floor}(\text{input_dimension} * \text{scale})$.

Syntax

```
struct DML_UPSAMPLE_2D_OPERATOR_DESC {
    const DML_TENSOR_DESC *InputTensor;
    const DML_TENSOR_DESC *OutputTensor;
    DML_SIZE_2D ScaleSize;
    DML_INTERPOLATION_MODE InterpolationMode;
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

ScaleSize

Type: **DML_SIZE_2D**

The value to scale by, as a 2-D size.

InterpolationMode

Type: **DML_INTERPOLATION_MODE**

The interpolation mode to use for the upsample. You can use the default value of **DML_INTERPOLATION_MODE_NEAREST_NEIGHBOR**.

Requirements

Header	directml.h

DML_VALUE_SCALE_2D_OPERATOR_DESC structure

5/27/2020 • 2 minutes to read • [Edit Online](#)

Describes a DirectML operator that performs an element-wise scale-and-bias function on the values in the input tensor. This operator is similar to using an ELEMENT_WISE_IDENTITY operator with a scale and bias, except that VALUE_SCALE_2D applies a different bias for each channel rather than a single bias for the entire tensor.

Syntax

```
struct DML_VALUE_SCALE_2D_OPERATOR_DESC {  
    const DML_TENSOR_DESC *InputTensor;  
    const DML_TENSOR_DESC *OutputTensor;  
    FLOAT Scale;  
    UINT ChannelCount;  
    const FLOAT *Bias;  
};
```

Members

InputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to read from.

OutputTensor

Type: **const DML_TENSOR_DESC***

A pointer to a constant **DML_TENSOR_DESC** containing the description of the tensor to write the results to.

Scale

Type: **FLOAT**

The scale to apply. You can use a default value of 1.0.

ChannelCount

Type: **UINT**

This field determines the size of the *Bias* array. This field must be set to either 1 or 3, and must also match the size of the C dimension of the input tensor.

Bias

Type: **const FLOAT***

A pointer to a constant array of **FLOAT** containing the bias term for each dimension of the input tensor.

Requirements

Header	directml.h
--------	------------

DMLCreateDevice function

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a DirectML device for a given Direct3D 12 device.

Syntax

```
HRESULT DMLCreateDevice(  
    ID3D12Device             *d3d12Device,  
    DML_CREATE_DEVICE_FLAGS   flags,  
    REFIID                   riid,  
    void                     **ppv  
) ;
```

Parameters

d3d12Device

Type: [ID3D12Device](#)*

A pointer to an [ID3D12Device](#) representing the Direct3D 12 device to create the DirectML device over. DirectML supports any D3D feature level, and Direct3D 12 devices created on any adapter, including WARP. However, not all features in DirectML may be available depending on the capabilities of the Direct3D 12 device. See [IDMLDevice::CheckFeatureSupport](#) for more information.

If the call to **DMLCreateDevice** is successful, the DirectML device maintains a strong reference to the supplied Direct3D 12 device.

flags

Type: [DML_CREATE_DEVICE_FLAGS](#)

A [DML_CREATE_DEVICE_FLAGS](#) value specifying additional device creation options.

riid

Type: [REFIID](#)

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

ppv

Return value

Type: [HRESULT](#)

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

To use the debug layers, developer mode must be enabled, and the DirectML debug layers must be installed. So, if the [DML_CREATE_DEVICE_FLAG_DEBUG](#) flag is specified in *flags* and either condition is not met, then **DMLCreateDevice** returns [DXGI_ERROR_SDK_COMPONENT_MISSING](#).

Requirements

Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

IDMLBindingTable interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Wraps a range of an application-managed descriptor heap, and is used by DirectML to create bindings for resources. To create this object, call [IDMLDevice::CreateBindingTable](#). The **IDMLBindingTable** interface inherits from [IDMLDeviceChild](#).

The binding table is created over a range of CPU and GPU descriptor handles. When an `IDMLBindingTable::Bind*` method is called, DirectML writes one or more descriptors into the range of CPU descriptors. When you use the binding table during a call to [IDMLCommandRecorder::RecordDispatch](#), DirectML binds the corresponding GPU descriptors to the pipeline.

The CPU and GPU descriptor handles aren't required to point to the same entries in a descriptor heap, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table.

It is your application's responsibility to perform correct synchronization between the CPU and GPU work that uses this binding table. For example, you must take care not to overwrite the bindings created by the binding table (for example, by calling `Bind*` again on the binding table, or by overwriting the descriptor heap manually) until all work using the binding table has completed execution on the GPU. In addition, since the binding table doesn't maintain a reference on the descriptor heap it writes into, you must not release the backing shader-visible descriptor heap until all work using that binding table has completed execution on the GPU.

The binding table is associated with exactly one dispatchable object (an operator initializer, or a compiled operator), and represents the bindings for that particular object. You can reuse a binding table by calling [IDMLBindingTable::Reset](#), however. Note that since the binding table doesn't own the descriptor heap itself, it is safe to call `Reset` and reuse the binding table for a different dispatchable object even before any outstanding executions have completed on the GPU.

The binding table doesn't keep strong references on any resources bound using it—your application must ensure that resources are not deleted while still in use by the GPU.

This object is not thread safe—your application must not call methods on the binding table simultaneously from different threads without synchronization.

Inheritance

The **IDMLBindingTable** interface inherits from the **IDMLDeviceChild** interface.

Methods

The **IDMLBindingTable** interface has these methods.

METHOD	DESCRIPTION
IDMLBindingTable::BindInputs	Binds a set of resources as input tensors.
IDMLBindingTable::BindOutputs	Binds a set of resources as output tensors.
IDMLBindingTable::BindPersistentResource	Binds a buffer as a persistent resource. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties .

METHOD	DESCRIPTION
IDMLBindingTable::BindTemporaryResource	Binds a buffer to use as temporary scratch memory. You can determine the required size of this buffer range by calling IDMLDispatchable::GetBindingProperties .
IDMLBindingTable::Reset	Resets the binding table to wrap a new range of descriptors, potentially for a different operator or initializer. This allows dynamic reuse of the binding table.

Requirements

Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#), [IDMLDeviceChild](#)

IDMLBindingTable::BindInputs method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Binds a set of resources as input tensors.

If binding for a compiled operator, the number of bindings must exactly match the number of inputs of the operator, including optional tensors. This can be determined from the operator description used to create the operator. If too many or too few bindings are provided, device removal will occur. For optional tensors, you may use [DML_BINDING_TYPE_NONE](#) to specify 'no binding'. Otherwise, the binding type must match the tensor type when the operator was created.

For operator initializers, input bindings are expected to be of type [DML_BINDING_TYPE_BUFFER_ARRAY](#) with one input binding per operator to initialize, supplied in the order that you specified the operators during creation or reset of the initializer. Each buffer array should have a size equal to the number of inputs of its corresponding operator to initialize. Input tensors that had the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set should be bound during initialize—otherwise, nothing should be bound for that tensor. If there is nothing to be bound as input for initialization of an operator (that is, there are no tensors with the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set) then you may supply `nullptr` or an empty [DML_BUFFER_ARRAY_BINDING](#) to indicate 'no binding'.

To unbind all input resources, supply a *rangeCount* of 0, and a value of `nullptr` for *bindings*.

If an input tensor has the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set, it may only be bound when executing an operator initializer. Otherwise, if the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag is not set, the opposite is true—the input tensor must not be bound when executing the initializer, but must be bound when executing the operator itself.

All buffers being bound as input must have heap type [D3D12_HEAP_TYPE_DEFAULT](#), except when the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag is set. If the [DML_TENSOR_FLAG OWNED_BY_DML](#) is set for a tensor that is being bound as input for an initializer, the buffer's heap type may be either [D3D12_HEAP_TYPE_DEFAULT](#) or [D3D12_HEAP_TYPE_UPLOAD](#).

Each binding is not required to point to a unique resource. It is legal, for example, to supply all inputs as suballocations from the same Direct3D 12 buffer resource. Ranges for inputs are permitted to overlap. No Direct3D 12 resource bound for input may simultaneously be bound for output, except if the operator explicitly permits in-place execution. If using in-place execution, the input and output tensors must have the exact same sizes, strides, and total tensor size. Additionally, the input and output bindings must have the exact same offset size.

Syntax

```
void BindInputs(  
    UINT             bindingCount,  
    const DML_BINDING_DESC *bindings  
>;
```

Parameters

`bindingCount`

Type: [UINT](#)

This parameter determines the size of the *bindings* array (if provided).

`bindings`

Type: `const DML_BINDING_DESC*`

An optional pointer to a constant array of `DML_BINDING_DESC` containing descriptions of the tensor resources to bind.

Return value

None

Requirements

Target Platform	Windows
Header	<code>directml.h</code>
Library	<code>DirectML.lib</code>
DLL	<code>DirectML.dll</code>

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

IDMLBindingTable::BindOutputs method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Binds a set of resources as output tensors.

If binding for a compiled operator, the number of bindings must exactly match the number of inputs of the operator, including optional tensors. This can be determined from the operator description used to create the operator. If too many or too few bindings are provided, device removal will occur. For optional tensors, you may use `DML_BINDING_TYPE_NONE`(/windows/desktop/api/directml/ne-directml-dml_binding_type) to specify 'no binding'. Otherwise, the binding type must match the tensor type when the operator was created.

For operator initializers, the output bindings are the persistent resources of each operator, supplied in the order the operators were given when creating or resetting the initializer. If a particular operator does not require a persistent resource, you should prove an empty binding in that slot.

To unbind all input resources, supply a `rangeCount` of 0, and a value of `nullptr` for `bindings`.

The writeable areas of two output tensors must not overlap with one another. The 'writeable area' of an output buffer being bound is defined as being the start offset of the buffer range, up to the `TotalTensorSizeInBytes` as specified in the tensors description.

All buffers being bound as output must have heap type `D3D12_HEAP_TYPE_DEFAULT`.

Syntax

```
void BindOutputs(  
    UINT             bindingCount,  
    const DML_BINDING_DESC *bindings  
>;
```

Parameters

`bindingCount`

Type: `UINT`

This parameter determines the size of the `bindings` array (if provided).

`bindings`

Type: `const DML_BINDING_DESC*`

An optional pointer to a constant array of `DML_BINDING_DESC` containing descriptions of the tensor resources to bind.

Return value

None

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

IDMLBindingTable::BindPersistentResource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Binds a buffer as a persistent resource. You can determine the required size of this buffer range by calling [IDMLDispatchable::GetBindingProperties](#).

If the binding properties for the operator specify a size of zero for the persistent resource, then you may supply `nullptr` to this method (which indicates no resource to bind). Otherwise, a binding of type `DML_BINDING_TYPE_BUFFER` must be supplied that is at least as large as the required `PersistentResourceSize` returned by [IDMLDispatchable::GetBindingProperties](#).

Unlike the temporary resource, the persistent resource's contents and lifetime must persist as long as the compiled operator does. That is, if an operator requires a persistent resource, then your application must supply it during initialization and subsequently also supply it to all future executes of the operator without modifying its contents.

The persistent resource is typically used by DirectML to store lookup tables or other long-lived data that is computed during initialization of an operator and reused on future executions of that operator.

As the persistent resource's data is opaque, once initialized it cannot be copied or moved to another buffer.

The persistent resource is only written to during initialization of an operator and is thereafter immutable; all subsequent executions are guaranteed not to write to the persistent resource.

The supplied buffer range to be bound as the persistent buffer must have its start offset aligned to `DML_PERSISTENT_BUFFER_ALIGNMENT`. The type of the heap underlying the buffer must be `D3D12_HEAP_TYPE_DEFAULT`.

Syntax

```
void BindPersistentResource(  
    const DML_BINDING_DESC *binding  
>;
```

Parameters

`binding`

Type: `const DML_BINDING_DESC*`

An optional pointer to a `DML_BINDING_DESC` containing the description of a tensor resource to bind.

Return value

None

Requirements

Target Platform	Windows

Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

IDMLBindingTable::BindTemporaryResource method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Binds a buffer to use as temporary scratch memory. You can determine the required size of this buffer range by calling [IDMLDispatchable::GetBindingProperties](#).

If the binding properties for the [IDMLDispatchable](#) specify a size of zero for the temporary resource, then you may supply `nullptr` to this method (which indicates no resource to bind). Otherwise, a binding of type [DML_BINDING_TYPE_BUFFER](#) must be supplied that is at least as large as the required [TemporaryResourceSize](#) returned by [IDMLDispatchable::GetBindingProperties](#).

The temporary resource is typically used as scratch memory during execution of an operator. The contents of a temporary resource need not be defined prior to execution. For example, DirectML doesn't require that you zero the contents of the temporary resource prior to binding or executing an operator.

You don't need to preserve the contents of the temporary buffer, and your application is free to overwrite or reuse its contents as soon as execution of an operator or initializer completes on the GPU. This is in contrast to a persistent resource, whose contents must be preserved and lifetime extended for the lifetime of the operator.

The supplied buffer range to be bound as the temporary buffer must have its start offset aligned to [DML_TEMPORARY_BUFFER_ALIGNMENT](#). The type of the heap underlying the buffer must be [D3D12_HEAP_TYPE_DEFAULT](#).

Syntax

```
void BindTemporaryResource(  
    const DML_BINDING_DESC *binding  
>;
```

Parameters

`binding`

Type: `const DML_BINDING_DESC*`

An optional pointer to a [DML_BINDING_DESC](#) containing the description of a tensor resource to bind.

Return value

None

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib

DLL	DirectML.dll
-----	--------------

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

IDMLBindingTable::Reset method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Resets the binding table to wrap a new range of descriptors, potentially for a different operator or initializer. This allows dynamic reuse of the binding table.

Resetting a binding table doesn't modify any previous bindings created by the table. Because of this, it is safe to reset the binding table immediately after supplying it to [IDMLCommandRecorder::RecordDispatch](#), even if that work has not yet completed execution on the GPU, so long as the underlying descriptors remain valid.

See [IDMLDevice::CreateBindingTable](#) for more information on the parameters supplied to this method.

Syntax

```
HRESULT Reset(  
    const DML_BINDING_TABLE_DESC *desc  
) ;
```

Parameters

desc

Type: [const DML_BINDING_TABLE_DESC*](#)

An optional pointer to a [DML_BINDING_TABLE_DESC](#) containing the binding table parameters. This may be `nullptr`, indicating an empty binding table.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns `S_OK`. Otherwise, it returns an [HRESULT](#) error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLBindingTable](#)

IDMLCommandRecorder interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Records dispatches of DirectML work into a Direct3D 12 command list. The **IDMLCommandRecorder** interface inherits from [IDMLDeviceChild](#).

The command recorder is a stateless object whose purpose is to record commands into a Direct3D 12 command list. DirectML doesn't create command lists, command allocators, nor command queues; nor does it directly submit any work for execution on the GPU. Instead, your application manages its own command lists and queues, and it uses the **IDMLCommandRecorder** to record work into its existing command lists. You're then responsible for executing the command list on a queue of your choice.

This object is thread-safe.

Inheritance

The **IDMLCommandRecorder** interface inherits from the [IDMLDeviceChild](#) interface.

Methods

The **IDMLCommandRecorder** interface has these methods.

METHOD	DESCRIPTION
IDMLCommandRecorder::RecordDispatch	Records execution of a dispatchable object (an operator initializer, or a compiled operator) onto a command list.

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLDeviceChild](#)

IDMLCommandRecorder::RecordDispatch method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Records execution of a dispatchable object (an operator initializer, or a compiled operator) onto a command list.

This method doesn't submit the execution to the GPU; it merely records it onto the command list. You are responsible for closing the command list and submitting it to the Direct3D 12 command queue.

Prior to execution of this call on the GPU, all resources bound must be in the [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#) state, or a state implicitly promotable to [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#), such as [D3D12_RESOURCE_STATE_COMMON](#). After this call completes, the resources remain in the [D3D12_RESOURCE_STATE_UNORDERED_ACCESS](#) state. The only exception to this is for upload heaps bound when executing an operator initializer and while one or more tensors has the [DML_TENSOR_FLAG OWNED_BY_DML](#) flag set. In that case, any upload heaps bound for input must be in the [D3D12_RESOURCE_STATE_GENERIC_READ](#) state and will remain in that state, as required by all upload heaps.

This method resets the following state on the command list.

- Compute root signature
- Pipeline state

No other command list state is modified.

Although this method takes a binding table representing the resources to bind to the pipeline, it doesn't set the descriptor heaps containing the descriptors themselves. Therefore, your application is responsible for calling [ID3D12GraphicsCommandList::SetDescriptorHeaps](#) to bind the correct descriptor heaps to the pipeline.

If [DML_EXECUTION_FLAG DESCRIPTORS_VOLATILE](#) was not set when compiling the operator, then all bindings must be set on the binding table before **RecordDispatch** is called, otherwise the behavior is undefined. Otherwise, if the [_DESCRIPTORS_VOLATILE](#) flag is set, binding of resources may be deferred until the Direct3D 12 command list is submitted to the command queue for execution.

This method acts logically like a call to [ID3D12GraphicsCommandList::Dispatch](#). As such, unordered access view (UAV) barriers are necessary to ensure correct ordering if there are data dependencies between dispatches. This method does not insert UAV barriers on input nor output resources. Your application must ensure that the correct UAV barriers are performed on any inputs if their contents depend on an upstream dispatch, and on any outputs if there are downstream dispatches that depend on those outputs.

This method doesn't hold references to any of the interfaces passed in. It is your responsibility to ensure that the [IDMLDispatchable](#) object is not released until all dispatches using it have completed execution on the GPU.

Syntax

```
void RecordDispatch(  
    ID3D12CommandList *commandList,  
    IDMLDispatchable  *dispatchable,  
    IDMLBindingTable *bindings  
>;
```

Parameters

`commandList`

Type: [ID3D12CommandList*](#)

A pointer to an [ID3D12CommandList](#) interface representing the command list to record the execution into. The command list must be open and must have type [D3D12_COMMAND_LIST_TYPE_DIRECT](#) or [D3D12_COMMAND_LIST_TYPE_COMPUTE](#).

`dispatchable`

Type: [IDMLDispatchable*](#)

A pointer to an [IDMLDispatchable](#) interface representing the object (an operator initializer, or a compiled operator) whose execution will be recorded into the command list.

`bindings`

Type: [IDMLBindingTable*](#)

A pointer to an [IDMLBindingTable](#) interface representing the bindings to use for executing the dispatchable object. If the [DML_EXECUTION_FLAG_DESCRIPTORS_VOLATILE](#) flag was not set, then you must fill out all required bindings, otherwise an error will result.

Return value

None

Requirements

Target Platform	Windows
Header	<code>directml.h</code>
Library	<code>DirectML.lib</code>
DLL	<code>DirectML.dll</code>

See also

[Binding in DirectML](#)

[IDMLCommandRecorder](#)

IDMLCompiledOperator interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a compiled, efficient form of an operator suitable for execution on the GPU. To create this object, call [IDMLDevice::CompileOperator](#). The **IDMLCompiledOperator** interface inherits from [IDMLDispatchable](#).

Unlike [IDMLOperator](#), compiled operators are "baked", and can be executed directly by the GPU. After an operator is compiled, you must initialize it exactly once before it can be executed. It's an error to initialize an operator more than once. Operator initializers are used to initialize compiled operators. You can use [IDMLCommandRecorder::RecordDispatch](#) to record the dispatch of an operator initializer which, when executed on the GPU, will initialize one or more operators.

In addition to input and output tensors, operators may require additional memory for execution. This additional memory must be provided by your application in the form of temporary and persistent resources.

A temporary resource is scratch memory that is only used during the execution of the operator, and doesn't need to persist after the call to [IDMLCommandRecorder::RecordDispatch](#) completes on the GPU. This means that your application may release or overwrite the temporary resource in between dispatches of the compiled operator. In contrast, the persistent resource must live at least until the last execute of the operator has completed on the GPU. Additionally, the contents of the persistent resource are opaque and must be preserved between executions of the operator.

The size of the temporary and persistent resources varies per operator. Call [IDMLDispatchable::GetBindingProperties](#) to query the required size, in bytes, of the persistent and temporary resources for this compiled operator. See [IDMLBindingTable::BindTemporaryResource](#) and [IDMLBindingTable::BindPersistentResource](#) for more information on binding temporary and persistent resources.

All methods on this interface are thread-safe.

Inheritance

The **IDMLCompiledOperator** interface inherits from the **IDMLDispatchable** interface.

Methods

The **IDMLCompiledOperator** interface has these methods.

METHOD	DESCRIPTION
--------	-------------

Requirements

Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#)

IDMLDispatchable

IDMLDebugDevice interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Controls the DirectML debug layers. The **IDMLDebugDevice** interface inherits from the [IUnknown](#) interface.

This interface may be retrieved from the [IDMLDevice](#) via [QueryInterface](#). This interface is available only if the DirectML device was created with the [DML_CREATE_DEVICE_FLAG_DEBUG](#) flag. DirectML sends messages to the [ID3D12InfoQueue](#) associated with the [ID3D12Device](#) passed in at [DMLCreateDevice](#); the Direct3D 12 info queue can be retrieved via [QueryInterface](#) on the [ID3D12Device](#).

This object is thread-safe.

Inheritance

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

Methods

The **IDMLDebugDevice** interface has these methods.

METHOD	DESCRIPTION
IDMLDebugDevice::SetMuteDebugOutput	Determine whether to mute DirectML from sending messages to the ID3D12InfoQueue .

Requirements

Target Platform	Windows
Header	directml.h

IDMLDebugDevice::SetMuteDebugOutput method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Determine whether to mute DirectML from sending messages to the [ID3D12InfoQueue](#).

Syntax

```
void SetMuteDebugOutput(  
    BOOL mute  
>);
```

Parameters

`mute`

Type: [BOOL](#)

If **TRUE**, DirectML is muted, and it will not send messages to the [ID3D12InfoQueue](#). If **FALSE**, DirectML is not muted, and it will send messages to the [ID3D12InfoQueue](#). The default value is **FALSE**.

Return value

None

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDebugDevice](#)

IDMLDevice interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

Represents a DirectML device, which is used to create operators, binding tables, command recorders, and other objects. The **IDMLDevice** interface inherits from [IDMLObject](#).

A DirectML device is always associated with exactly one underlying Direct3D 12 device. All objects created by the DirectML device maintain a strong reference to their parent device. Unlike the Direct3D 12 device, the DML device is not a singleton. Therefore, it's possible to create multiple DirectML devices over the same Direct3D 12 device. However, this isn't recommended as the DirectML device has no mutable state, so there's little advantage to creating multiple DML devices over the same Direct3D 12 device.

This object is thread-safe.

Inheritance

The **IDMLDevice** interface inherits from the [IDMLObject](#) interface.

Methods

The **IDMLDevice** interface has these methods.

METHOD	DESCRIPTION
IDMLDevice::CheckFeatureSupport	Gets information about the optional features and capabilities that are supported by the DirectML device.
IDMLDevice::CompileOperator	Compiles an operator into an object that can be dispatched to the GPU.
IDMLDevice::CreateBindingTable	Creates a binding table, which is an object that can be used to bind resources (such as tensors) to the pipeline.
IDMLDevice::CreateCommandRecorder	Creates a DirectML command recorder.
IDMLDevice::CreateOperator	Creates a DirectML operator.
IDMLDevice::CreateOperatorInitializer	Creates an object that can be used to initialize compiled operators.
IDMLDevice::Evict	Evicts one or more pageable objects from GPU memory. Also see IDMLDevice::MakeResident .
IDMLDevice::GetDeviceRemovedReason	Retrieves the reason that the DirectML device was removed.
IDMLDevice::GetParentDevice	Retrieves the Direct3D 12 device that was used to create this DirectML device.
IDMLDevice::MakeResident	Causes one or more pageable objects to become resident in GPU memory. Also see IDMLDevice::Evict .

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLObject](#)

IDMLDevice::CheckFeatureSupport method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets information about the optional features and capabilities that are supported by the DirectML device.

Syntax

```
HRESULT CheckFeatureSupport(  
    DML_FEATURE feature,  
    UINT        featureQueryDataSize,  
    const void  *featureQueryData,  
    UINT        featureSupportDataSize,  
    void       *featureSupportData  
) ;
```

Parameters

`feature`

Type: [DML_FEATURE](#)

A constant from the [DML_FEATURE](#) enumeration describing the feature(s) that you want to query for support.

`featureQueryDataSize`

Type: [UINT](#)

The size of the structure pointed to by the *featureQueryData* parameter, if provided, otherwise 0.

`featureQueryData`

Type: [const void*](#)

An optional pointer to a query structure that corresponds to the value of the *feature* parameter. To determine the corresponding query type for each constant, see [DML_FEATURE](#).

`featureSupportDataSize`

Type: [UINT](#)

The size of the structure pointed to by the *featureSupportData* parameter.

`featureSupportData`

Type: [void*](#)

A pointer to a support data structure that corresponds to the value of the *feature* parameter. To determine the corresponding support data type for each constant, see [DML_FEATURE](#).

Return value

Type: [HRESULT](#)

If this method succeeds, it returns [S_OK](#). Otherwise, it returns [DXGI_ERROR_UNSUPPORTED](#) if the [DML_FEATURE](#) is unrecognized or unsupported, and [E_INVALIDARG](#) if the parameters are incorrect.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::CompileOperator method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Compiles an operator into an object that can be dispatched to the GPU.

A compiled operator represents the efficient, baked form of an operator suitable for execution on the GPU. A compiled operator holds state (such as shaders and other objects) required for execution. Because a compiled operator implements the [IDMLPageable](#) interface, you're able to evict one from GPU memory if you wish. See [IDMLDevice::Evict](#) and [IDMLDevice::MakeResident](#) for more info.

The compiled operator maintains a strong reference to the supplied [IDMLOperator](#) pointer.

Syntax

```
HRESULT CompileOperator(
    IDMLOperator     *op,
    DML_EXECUTION_FLAGS flags,
    REFIID           riid,
    void             **ppv
);
```

Parameters

`op`

Type: [IDMLOperator](#)*

The operator (created with [IDMLDevice::CreateOperator](#)) to compile.

`flags`

Type: [DML_EXECUTION_FLAGS](#)

Any flags to control the execution of this operator.

`riid`

Type: [REFIID](#)

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

`ppv`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the compiled operator. This is the address of a pointer to an [IDMLCompiledOperator](#), representing the compiled operator created.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::CreateBindingTable method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a binding table, which is an object that can be used to bind resources (such as tensors) to the pipeline.

The binding table wraps a range of an application-managed descriptor heap using the provided descriptor handles and count. Binding tables are used by DirectML to manage the binding of resources by writing descriptors into the descriptor heap at the offset specified by the **CPUDescriptorHandle**, and binding those descriptors to the pipeline using the descriptors at the offset specified by the **GPUDescriptorHandle**. The order in which DirectML writes descriptors into the heap is unspecified, so your application must take care not to overwrite the descriptors wrapped by the binding table.

The supplied CPU and GPU descriptor handles may come from different heaps, however it is then your application's responsibility to ensure that the entire descriptor range referred to by the CPU descriptor handle is copied into the range referred to by the GPU descriptor handle prior to execution using this binding table.

The descriptor heap from which the handles are supplied must have type

[D3D12_DESCRIPTOR_HEAP_TYPE_CBV_SRV_UAV](#). Additionally, the heap referred to by the

GPUDescriptorHandle must be a shader-visible descriptor heap.

You must not delete the heap referred to by the GPU descriptor handle until all work referencing it has completed execution on the GPU. You may, however, reset or release the binding table itself as soon as the dispatch has been recorded into the command list. Similar to the relationship between [ID3D12CommandList](#) and [ID3D12CommandAllocator](#), the [IDMLBindingTable](#) doesn't own the underlying memory referenced by the descriptor handles. Rather, the [ID3D12DescriptorHeap](#) does. Therefore, you're permitted to reset or release a DirectML binding table before work using the binding table has completed execution on the GPU.

Syntax

```
HRESULT CreateBindingTable(  
    const DML_BINDING_TABLE_DESC *desc,  
    REFIID                   riid,  
    void                      **ppv  
) ;
```

Parameters

`desc`

Type: [const DML_BINDING_TABLE_DESC*](#)

An optional pointer to a [DML_BINDING_TABLE_DESC](#) containing the binding table parameters. This may be `nullptr`, indicating an empty binding table.

`riid`

Type: [REFIID](#)

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

`ppv`

Type: **void****

A pointer to a memory block that receives a pointer to the binding table. This is the address of a pointer to an [IDMLBindingTable](#), representing the binding table created.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLDevice](#)

IDMLDevice::CreateCommandRecorder method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a DirectML command recorder.

A command recorder allows your application to record the initialization and execution of compiled operators into existing Direct3D 12 command lists. The command recorder is a stateless object: it does not own command lists or operators, nor does it execute any GPU work. Instead, it merely records the commands necessary for dispatching initialization or execution into an application-supplied command list. Your application is then responsible for submitting the execution of that command list to the Direct3D 12 command queue.

Syntax

```
HRESULT CreateCommandRecorder(  
    REFIID riid,  
    void    **ppv  
)
```

Parameters

`riid`

Type: [REFIID](#)

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

`ppv`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the command recorder. This is the address of a pointer to an [IDMLCommandRecorder](#), representing the command recorder created.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::CreateOperator method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates a DirectML operator.

In DirectML, an operator represents an abstract bundle of functionality, which can be compiled into a form suitable for execution on the GPU. Operator objects cannot be executed directly; they must first be compiled into an [IDMLCompiledOperator](#).

Syntax

```
HRESULT CreateOperator(
    const DML_OPERATOR_DESC *desc,
    REFIID                 riid,
    void                   **ppv
);
```

Parameters

`desc`

Type: [const DML_OPERATOR_DESC*](#)

The description of the operator to be created.

`riid`

Type: [REFIID](#)

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

`ppv`

Type: [void**](#)

A pointer to a memory block that receives a pointer to the operator. This is the address of a pointer to an [IDMLOperator](#), representing the operator created.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h

Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::CompileOperator](#)

IDMLDevice::CreateOperatorInitializer method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Creates an object that can be used to initialize compiled operators.

Once compiled, an operator must be initialized exactly once on the GPU before it can be executed. The operator initializer holds the state necessary for initialization of one or more target compiled operators.

Once instantiated, dispatch of the operator initializer can be recorded in a command list via [IDMLCommandRecorder::RecordDispatch](#). After execution completes on the GPU, all compiled operators that are targets of the initializer enter the initialized state.

An operator initializer can be reused to initialize different sets of compiled operators. See [IDMLOperatorInitializer::Reset](#) for more info.

An operator initializer can be created with no target operators. Executing such an initializer is a no-op. Creating an operator initializer with no target operators may be useful if you wish to create an initializer up-front, but don't yet know which operators it will be used to initialize. [IDMLOperatorInitializer::Reset](#) can be used to reset which operators to target.

Syntax

```
HRESULT CreateOperatorInitializer(
    UINT             operatorCount,
    IDMLCompiledOperator * const *operators,
    REFIID          riid,
    void            **ppv
);
```

Parameters

operatorCount

Type: [UINT](#)

This parameter determines the number of elements in the array passed in the *operators* parameter.

operators

Type: [IDMLCompiledOperator*](#)

An optional pointer to a constant array of [IDMLCompiledOperator](#) pointers containing the set of operators that this initializer will target. Upon execution of the initializer, the target operators become initialized. This array may be null or empty, indicating that the initializer has no target operators.

riid

Type: [REFIID](#)

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

ppv

Type: [void**](#)

A pointer to a memory block that receives a pointer to the operator initializer. This is the address of a pointer to an [IDMLOperatorInitializer](#), representing the operator initializer created.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::Evict method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Evicts one or more pageable objects from GPU memory. Also see [IDMLDevice::MakeResident](#).

Syntax

```
HRESULT Evict(  
    UINT          count,  
    IDMLPageable * const *ppObjects  
) ;
```

Parameters

count

Type: [UINT](#)

This parameter determines the number of elements in the array passed in the *ppObjects* parameter.

ppObjects

Type: [IDMLPageable*](#)

A pointer to a constant array of [IDMLPageable](#) pointers containing the pageable objects to evict from GPU memory.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::MakeResident](#)

IDMLDevice::GetDeviceRemovedReason method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the reason that the DirectML device was removed.

Syntax

```
HRESULT GetDeviceRemovedReason();
```

Parameters

This method has no parameters.

Return value

Type: [HRESULT](#)

An [HRESULT](#) containing the reason that the device was removed, or [S_OK](#) if the device has not been removed.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::GetParentDevice method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the Direct3D 12 device that was used to create this DirectML device.

Syntax

```
HRESULT GetParentDevice(  
    REFIID riid,  
    void    **ppv  
) ;
```

Parameters

`riid`

Type: [REFIID](#)

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

`ppv`

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 device.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

IDMLDevice::MakeResident method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Causes one or more pageable objects to become resident in GPU memory. Also see [IDMLDevice::Evict](#).

Syntax

```
HRESULT MakeResident(  
    UINT          count,  
    IDMLPageable * const *ppObjects  
) ;
```

Parameters

count

Type: **UINT**

This parameter determines the number of elements in the array passed in the *ppObjects* parameter.

ppObjects

Type: [IDMLPageable*](#)

A pointer to a constant array of [IDMLPageable](#) pointers containing the pageable objects to make resident in GPU memory.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDevice](#)

[IDMLDevice::Evict](#)

IDMLDeviceChild interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

An interface implemented by all objects created from the DirectML device. The **IDMLDeviceChild** interface inherits from [IDMLObject](#).

Inheritance

The **IDMLDeviceChild** interface inherits from the **IDMLObject** interface.

Methods

The **IDMLDeviceChild** interface has these methods.

METHOD	DESCRIPTION
IDMLDeviceChild::GetDevice	Retrieves the DirectML device that was used to create this object.

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLObject](#)

IDMLDeviceChild::GetDevice method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the DirectML device that was used to create this object.

Syntax

```
HRESULT GetDevice(  
    REFIID riid,  
    void    **ppv  
) ;
```

Parameters

`riid`

Type: [REFIID](#)

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

`ppv`

Type: [void**](#)

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

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLDeviceChild](#)

IDMLDispatchable interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Implemented by objects that can be recorded into a command list for dispatch on the GPU, using [IDMLCommandRecorder::RecordDispatch](#). The **IDMLDispatchable** interface inherits from [IDMLPageable](#).

This interface is implemented by [IDMLCompiledOperator](#) and [IDMLOperatorInitializer](#).

Inheritance

The IDMLDispatchable interface inherits from the IDMLPageable interface.

Methods

The IDMLDispatchable interface has these methods.

METHOD	DESCRIPTION
IDMLDispatchable::GetBindingProperties	Retrieves the binding properties for a dispatchable object (an operator initializer, or a compiled operator).

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLPageable](#)

IDMLDispatchable::GetBindingProperties method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Retrieves the binding properties for a dispatchable object (an operator initializer, or a compiled operator). The binding properties value contains the required size of the binding table in descriptors, as well as the required size in bytes of the temporary and persistent resources required to execute this object.

When called on an operator initializer, the binding properties of the object may be different if retrieved both before and after a call to [IDMLOperatorInitializer::Reset](#).

Syntax

```
DML_BINDING_PROPERTIES GetBindingProperties();
```

Parameters

This method has no parameters.

Return value

Type: [DML_BINDING_PROPERTIES](#)

A [DML_BINDING_PROPERTIES](#) value containing binding properties.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[Binding in DirectML](#)

[IDMLDispatchable](#)

IDMLObject interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

An interface from which [IDMLDevice](#) and [IDMLDeviceChild](#) inherit directly (and all other interfaces, indirectly). Consequently, it provides methods common to all DirectML interfaces, specifically methods to associate private data, and to annotate object names. The [IDMLObject](#) interface inherits from the [IUnknown](#) interface.

Inheritance

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

Methods

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

METHOD	DESCRIPTION
IDMLObject::GetPrivateData	Gets application-defined data from a DirectML device object.
IDMLObject::SetName	Associates a name with the DirectML device object. This name is for use in debug diagnostics and tools.
IDMLObject::SetPrivateData	Sets application-defined data to a DirectML device object, and associates that data with an application-defined GUID.
IDMLObject::SetPrivateDataInterface	Associates an IUnknown-derived interface with the DirectML device object, and associates that interface with an application-defined GUID.

Requirements

Target Platform	Windows
Header	directml.h

IDMLObject::GetPrivateData method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Gets application-defined data from a DirectML device object. This method is thread-safe.

Syntax

```
HRESULT GetPrivateData(  
    REFGUID guid,  
    UINT     *dataSize,  
    void     *data  
) ;
```

Parameters

guid

Type: [REFGUID](#)

The GUID that is associated with the data.

dataSize

Type: [UINT*](#)

A pointer to a variable that on input contains the size, in bytes, of the buffer that *data* points to, and on output contains the size, in bytes, of the amount of data that **GetPrivateData** retrieved.

data

Type: [void*](#)

A pointer to a memory block that receives the data from the device object if *dataSize* points to a value that specifies a buffer large enough to hold the data.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

IDMLObject::SetName method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Associates a name with the DirectML device object. This name is for use in debug diagnostics and tools. This method is thread-safe.

Syntax

```
HRESULT SetName(  
    PCWSTR name  
)
```

Parameters

name

Type: [PCWSTR](#)

A NULL-terminated **UNICODE** string that contains the name to associate with the DirectML device object.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

IDMLObject::SetPrivateData method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Sets application-defined data to a DirectML device object, and associates that data with an application-defined GUID. This method is thread-safe.

Syntax

```
HRESULT SetPrivateData(  
    REFGUID    guid,  
    UINT       dataSize,  
    const void *data  
) ;
```

Parameters

guid

Type: [REFGUID](#)

The GUID to associate with the data.

dataSize

Type: [UINT](#)

The size in bytes of the data.

data

Type: [const void*](#)

A pointer to a memory block that contains the data to be stored with this DirectML device object. If *data* is **NULL**, then *dataSize* must be 0, and any data that was previously associated with the GUID specified in *guid* will be destroyed.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib

DLL	DirectML.dll
-----	--------------

See also

[IDMLObject](#)

IDMLObject::SetPrivateDataInterface method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Associates an [IUnknown](#)-derived interface with the DirectML device object, and associates that interface with an application-defined GUID. This method is thread-safe.

Syntax

```
HRESULT SetPrivateDataInterface(  
    REFGUID guid,  
    IUnknown *data  
) ;
```

Parameters

`guid`

Type: [REFGUID](#)

The GUID to associate with the interface.

`data`

Type: `const IUnknown*`

A pointer to the [IUnknown](#)-derived interface to be associated with the device object.

Return value

Type: [HRESULT](#)

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

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLObject](#)

IDMLOperator interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a DirectML operator. The **IDMLOperator** interface inherits from [IDMLDeviceChild](#).

Inheritance

The **IDMLOperator** interface inherits from the [IDMLDeviceChild](#) interface.

Methods

The **IDMLOperator** interface has these methods.

METHOD	DESCRIPTION
--------	-------------

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLDeviceChild](#)

IDMLOperatorInitializer interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Represents a specialized object whose purpose is to initialize compiled operators. To create an instance of this object, call [IDMLDevice::CreateOperatorInitializer](#). The **IDMLOperatorInitializer** interface inherits from [IDMLDispatchable](#).

An operator initializer is associated with one or more compiled operators, which are the targets for initialization. You can record operator initialization onto a command list using [IDMLCommandRecorder::RecordDispatch](#). When the initialization completes execution on the GPU, all of the target operators enter the initialized state. You must initialize all operators exactly once before they can be executed.

Inheritance

The **IDMLOperatorInitializer** interface inherits from the [IDMLDispatchable](#) interface.

Methods

The **IDMLOperatorInitializer** interface has these methods.

METHOD	DESCRIPTION
IDMLOperatorInitializer::Reset	Resets the initializer to handle initialization of a new set of operators.

Remarks

Operator initializers are reusable: once an instance has been used to initialize a set of operators, you can reset it with a different set of compiled operators as targets.

When executing an initializer, the expected bindings are as follows:

- Inputs should be one buffer array binding for each target operator, in the order that you originally specified the operators when creating or resetting the initializer. Each buffer array binding itself should have a size equal to the inputs of its respective operator. Alternatively, you may specify NONE for a binding to bind no inputs for initialization of that target operator.
- Outputs should be the persistent resources for each target operator, in the order that you originally specified the operators when creating or resetting the initializer.
- As with any dispatchable object (an operator initializer, or a compiled operator), the initializer may require a temporary resource. Call [IDMLDispatchable::GetBindingProperties](#) to determine the required size of the temporary resource.
- Operator initializers don't ever require persistent resources. Therefore, calling [IDMLDispatchable::GetBindingProperties](#) on an operator initializer always returns a **PersistentResourceSize** of 0.

The operator initializer itself doesn't need to be initialized—GPU initialization only applies to compiled operators.

Requirements

Target Platform	Windows
Header	directml.h

See also

[Binding in DirectML](#)

[IDMLDispatchable](#)

IDMLOperatorInitializer::Reset method

5/27/2020 • 2 minutes to read • [Edit Online](#)

Resets the initializer to handle initialization of a new set of operators.

You may use an initializer only to initialize a fixed set of operators, which are provided either during creation ([IDMLDevice::CreateOperatorInitializer](#)), or when the initializer is reset. Resetting the initializer allows your application to reuse an existing initializer object to initialize a new set of operators.

You must not call **Reset** until all outstanding work using the initializer has completed execution on the GPU.

This method is not thread-safe.

Syntax

```
HRESULT Reset(  
    UINT             operatorCount,  
    IDMLCompiledOperator * const *operators  
) ;
```

Parameters

operatorCount

Type: **UINT**

This parameter determines the number of elements in the array passed in the *operators* parameter.

operators

Type: [IDMLCompiledOperator*](#)

An optional pointer to a constant array of [IDMLCompiledOperator](#) pointers containing the operators that the initializer should initialize.

Return value

Type: [HRESULT](#)

If this method succeeds, it returns **S_OK**. Otherwise, it returns an **HRESULT** error code.

Requirements

Target Platform	Windows
Header	directml.h
Library	DirectML.lib
DLL	DirectML.dll

See also

[IDMLOperatorInitializer](#)

IDMLPageable interface

5/27/2020 • 2 minutes to read • [Edit Online](#)

Implemented by objects that can be evicted from GPU memory, and hence that can be supplied to [IDMLDevice::Evict](#) and [IDMLDevice::MakeResident](#). The **IDMLOperator** interface inherits from [IDMLDeviceChild](#).

Inheritance

The IDMLPageable interface inherits from the IDMLDeviceChild interface.

Methods

The **IDMLPageable** interface has these methods.

METHOD	DESCRIPTION
--------	-------------

Requirements

Target Platform	Windows
Header	directml.h

See also

[IDMLDeviceChild](#)

windows.graphics.holographic.interop.h header

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

TITLE	DESCRIPTION
graphics::holographic::interop::IHolographicCameraInterop	Extends <code>HolographicCamera</code> 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 <code>HolographicCameraRenderingParameters</code> class for applications that use Direct3D 12 for holographic rendering.
graphics::holographic::interop::IHolographicQuadLayerInterop	A nano-COM interface that allows COM interop with the <code>HolographicQuadLayer</code> 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 <code>HolographicQuadLayerUpdateParameters</code> class for applications that use Direct3D 12 for holographic rendering.

IHolographicCameraInterop interface

6/30/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
IHolographicCameraInterop::AcquireDirect3D12BufferResource	Acquires a Direct3D 12 buffer resource.
IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout	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	Creates a Direct3D 12 resource for use as a content buffer for the camera, with optional hardware protection.
IHolographicCameraInterop::UnacquireDirect3D12BufferResource	Un-acquires a Direct3D 12 buffer resource.

Remarks

To use this interface in C++/WinRT, QueryInterface for the IHolographicCameraInterop interface from the [HolographicCamera](#) object.

```
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.

```
winrt::com_ptr<IHolographicCameraInterop> spCameraInterop =  
    m_holographicCamera.as<IHolographicCameraInterop>();  
winrt::check_hresult(  
    spCameraInterop->CreateDirect3D12BackBufferResource(  
        spD3D12Device.get(),  
        &bufferDesc,  
        m_spD3D12BackBuffer[bufferSlot].put()));
```

Requirements

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

IHolographicCameraInterop::AcquireDirect3D12BufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraInterop::AcquireDirect3D12BufferResourceWithTimeout method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraInterop::CreateDirect3D12BackBufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraInterop::CreateDirect3D12HardwareProtectedBackBufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraInterop::UnacquireDirect3D12BufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraRenderingParametersInterop interface

5/13/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource	Commits a Direct3D 12 buffer for presentation on outputs associated with the HolographicCamera .
IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData	Commits a Direct3D 12 buffer for presentation on outputs associated with the HolographicCamera .

Remarks

To use this interface in [C++/WinRT](#), retrieve the [HolographicCameraRenderingParameters](#) object from the [HolographicFrame](#), and then [QueryInterface](#) for the **IHolographicCameraRenderingParametersInterop** interface.

```
auto holographicCameraRenderingParameters { holographicFrame.GetRenderingParameters(m_cameraPose) };
winrt::com_ptr<IHolographicCameraRenderingParametersInterop> holographicCameraRenderingParametersInterop
{
    holographicCameraRenderingParameters.as<IHolographicCameraRenderingParametersInterop>();
};
```

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.

```
auto holographicCameraRenderingParameters =
    holographicFrame->GetRenderingParameters(m_cameraPose);
Microsoft::WRL::ComPtr<IHolographicCameraRenderingParametersInterop>
    holographicCameraRenderingParametersInterop;
{
    Microsoft::WRL::ComPtr<IIInspectable> iInspectable = reinterpret_cast<IIInspectable*>
(holographicCameraRenderingParameters);
    DX::ThrowIfFailed(iInspectable.As(&holographicCameraRenderingParametersInterop));
}
```

Requirements

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

IHolographicCameraRenderingParametersInterop::CommitDirect3D12Resource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicCameraRenderingParametersInterop::CommitDirect3D12ResourceWithDepthData method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicQuadLayerInterop interface

5/13/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource	Acquires a Direct3D 12 buffer resource.
IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout	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	Creates a Direct3D 12 resource for use as a content buffer for the camera, with optional hardware protection.
IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource	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.

```
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 **IIInspectable***. Then QueryInterface for the **IHolographicQuadLayerInterop** interface from the **IIInspectable** pointer.

```
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

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

IHolographicQuadLayerInterop::AcquireDirect3D12BufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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](#)

IHolographicQuadLayerInterop::AcquireDirect3D12BufferResourceWithTimeout method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

[AcquireDirect3D12BufferResource](#)

IHolographicQuadLayerInterop::CreateDirect3D12ContentBufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicQuadLayerInterop::CreateDirect3D12HardwareProtectedContentBufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicQuadLayerInterop::UnacquireDirect3D12BufferResource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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

IHolographicQuadLayerUpdateParametersInterop interface

5/13/2020 • 2 minutes to read • [Edit Online](#)

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.

METHOD	DESCRIPTION
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<IIInspectable> iInspectable = reinterpret_cast<IIInspectable*>(quadLayerParameters);
    DX::ThrowIfFailed(iInspectable.As(&quadLayerParamsInterop));
}
```

Requirements

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

IHolographicQuadLayerUpdateParameters::CommitDirect3D12Resource method

5/13/2020 • 2 minutes to read • [Edit Online](#)

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

```
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

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